



NOVA
IMS

Information
Management
School

MGI

Mestrado em Gestão de Informação

Master Program in Information Management

AUGMENTING DATA WAREHOUSING ARCHITECTURES WITH HADOOP

Henrique José Rosa Dias

Dissertation presented as partial requirement for obtaining
the Master's degree in Information Management

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

AUGMENTING DATA WAREHOUSING ARCHITECTURES WITH HADOOP

by

Henrique Dias

Dissertation presented as partial requirement for obtaining the Master's degree in Information Management, with a specialization in Information Systems and Technologies Management

Advisor: Professor Roberto Henriques, PhD.

September 2017

DEDICATION

É por ti, para ti e especialmente graças a ti! Obrigado sempre, mãe.

ACKNOWLEDGEMENTS

There were times when it seemed impossible, after a while it just became hard but, in the end, it was done. Looking back now, and despite all the challenges and difficulties, I would most certainly do it again!

If it is true that a big part of this journey is a lonely one, it is also true that it would not be the same if I did not have the help of the people that, somewhere along the road, came and pushed me forward. This list is extensive but I would like to use this opportunity to convey some special messages of gratitude.

Firstly, I would like to express my sincere gratitude to my advisor, professor Roberto Henriques, not only for his advices, support and very thorough review of this dissertation, but also because it was greatly due to his Big Data Applications class that my interest for the Big Data world began to materialize, and, ultimately, this led to my thesis subject choice. Most probably without his class this dissertation would never have happened.

I would also like to thank my manager at Xerox, Simon Hawkins, for his flexibility and understanding throughout this journey that, at times, revealed to be difficult to separate from my professional obligations, and, of course, for his precious help in overcoming my constant doubts related to writing in English.

Last, but not the least, a big *spasibo* to Daria Simonova, whom I met thanks to Nova IMS, for her patience, keen eye, and the courage to challenge my ideas whenever she felt she had to, during her review of this dissertation.

ABSTRACT

As the volume of available data increases exponentially, traditional data warehouses struggle to transform this data into actionable knowledge. Data strategies that include the creation and maintenance of data warehouses have a lot to gain by incorporating technologies from the Big Data's spectrum. Hadoop, as a transformation tool, can add a theoretical infinite dimension of data processing, feeding transformed information into traditional data warehouses that ultimately will retain their value as central components in organizations' decision support systems.

This study explores the potentialities of Hadoop as a data transformation tool in the setting of a traditional data warehouse environment. Hadoop's execution model, which is oriented for distributed parallel processing, offers great capabilities when the amounts of data to be processed require the infrastructure to expand. Horizontal scalability, which is a key aspect in a Hadoop cluster, will allow for proportional growth in processing power as the volume of data increases.

Through the use of a Hive on Tez, in a Hadoop cluster, this study transforms television viewing events, extracted from Ericsson's Mediaroom Internet Protocol Television infrastructure, into pertinent audience metrics, like Rating, Reach and Share. These measurements are then made available in a traditional data warehouse, supported by a traditional Relational Database Management System, where they are presented through a set of reports.

The main contribution of this research is a proposed augmented data warehouse architecture where the traditional ETL layer is replaced by a Hadoop cluster, running Hive on Tez, with the purpose of performing the heaviest transformations that convert raw data into actionable information. Through a typification of the SQL statements, responsible for the data transformation processes, we were able to understand that Hadoop, and its distributed processing model, delivers outstanding performance results associated with the analytical layer, namely in the aggregation of large data sets.

Ultimately, we demonstrate, empirically, the performance gains that can be extracted from Hadoop, in comparison to an RDBMS, regarding speed, storage usage and scalability potential, and suggest how this can be used to evolve data warehouses into the age of Big Data.

KEYWORDS

Big Data; Hadoop; Hive; Tez; Data Warehousing; ETL; Television Audience Measurements

INDEX

1. Introduction.....	1
1.1. Background and problem identification.....	1
1.2. Study objectives.....	2
1.3. Study relevance and importance	3
1.4. Document structure	4
2. Theoretical framework	5
2.1. Introduction.....	5
2.2. Relational databases	6
2.3. Structured Query Language.....	8
2.4. Relational Database Management Systems.....	10
2.5. Data Warehousing.....	11
2.5.1. ETL/ELT	13
2.5.2. Dimensional modeling.....	14
2.5.3. DW 2.0.....	16
2.6. Big Data.....	17
2.7. Hadoop	18
2.7.1. HDFS	20
2.7.2. Map-Reduce	22
2.7.3. YARN	25
2.7.4. Tez	26
2.7.5. Hive.....	29
2.7.6. Impala	31
2.7.7. The SQL cycle.....	33
2.8. Hybrid approach	33
2.9. IPTV.....	34
2.10. Television audience measurements.....	36
2.11. Summary.....	37
3. Methodology	38
3.1. Introduction.....	38
3.2. Research design.....	38
3.2.1. Research philosophy	39
3.2.2. Research approach	40
3.2.3. Research strategy	41

3.2.4. Time horizon.....	43
3.2.5. Data collection.....	43
3.3. Summary.....	44
4. Design and development.....	45
4.1. Introduction.....	45
4.2. Problem description	45
4.3. Data warehouse architecture	46
4.4. Source data.....	48
4.4.1. Inventory information	48
4.4.2. Subscriber events	49
4.4.3. Source files	50
4.4.4. Data preparation	51
4.5. Data warehouse design	52
4.5.1. Staging Area tables.....	53
4.5.2. Inventory tables.....	54
4.5.3. Support tables	54
4.5.4. Fact tables	54
4.5.5. Aggregation tables.....	55
4.5.6. Transformation processes.....	55
4.6. Environment	59
4.7. RDBMS implementation	60
4.7.1. RDBMS infrastructure.....	60
4.7.2. RDBMS configuration	60
4.7.3. Data model design.....	62
4.7.4. Data transformation	65
4.8. Hadoop cluster implementation	66
4.8.1. Hadoop cluster infrastructure.....	67
4.8.2. Hadoop cluster configuration.....	68
4.8.3. Data model design.....	70
4.8.4. Data transformation	71
4.9. Summary.....	72
5. Evaluation	73
5.1. Introduction.....	73
5.2. Performance	73
5.2.1. Subscriber events transformation.....	74

5.2.2.Subscriber events segmentation	78
5.2.3.Subscriber events aggregation	79
5.3. Scalability	80
5.4. Storage.....	83
5.5. Data validation	84
5.6. Visualization.....	85
5.6.1.Reporting	85
5.6.2.Performance	86
5.7. Summary.....	87
6. Results and discussion	88
6.1. Introduction.....	88
6.2. Performance	88
6.2.1.Batch processing.....	88
6.2.2.Interactive querying	89
6.3. Scalability.....	89
6.4. Storage.....	91
6.5. Architectures	92
6.6. Summary.....	93
7. Conclusions.....	94
7.1. Key findings	94
7.2. Research question and established objectives	95
7.3. Main contributions	96
7.4. Limitations and recommendations for future works	96
8. Bibliography.....	98
9. Appendix.....	105
Appendix A. Mediaroom event types.....	105
Appendix A.1. Channel Tune.....	105
Appendix A.2. Box Power.....	105
Appendix A.3. Trick State	105
Appendix A.4. Program Watched	106
Appendix A.5. DVR Start Recording	106
Appendix A.6. DVR Abort Recording.....	107
Appendix A.7. DVR Playback Recording.....	107
Appendix A.8. DVR Schedule Recording	107
Appendix A.9. DVR Delete Recording	108

Appendix A.10. DVR Cancel Recording	108
Appendix B. Data preparation scripts	110
Appendix C. Data Dictionary.....	111
Appendix C.1. Staging Area tables	111
Appendix C.2. Inventory tables.....	115
Appendix C.3. Support tables	119
Appendix C.4. Fact tables.....	119
Appendix C.5. Aggregation tables.....	122
Appendix D. Data warehouse RDMBS implementation	123
Appendix D.1. Storage options analysis.....	123
Appendix D.2. Tablespaces	124
Appendix D.3. Directories	125
Appendix D.4. Staging Area tables.....	126
Appendix D.5. Support tables	133
Appendix D.6. Inventory tables	133
Appendix D.7. Fact tables	141
Appendix D.8. Aggregation tables	153
Appendix D.9. Support procedures	156
Appendix D.10. Execution plans	158
Appendix E. Hadoop cluster implementation	161
Appendix E.1. Storage options analysis	161
Appendix E.2. Staging Area tables	163
Appendix E.3. Support tables.....	167
Appendix E.4. Inventory tables	167
Appendix E.5. Fact tables.....	174
Appendix E.6. Aggregation tables.....	183
Appendix E.7. Transformation processes DAGs	186
Appendix F. Performance execution statistics	189
Appendix F.1. Channel Tune	189
Appendix F.2. Program Watched.....	189
Appendix F.3. DVR Events.....	190
Appendix F.4. Event Segmentation.....	191
Appendix F.5. Audiences Aggregation	192
Appendix G. Scalability execution statistics	193
Appendix G.1. Channel Tune	193

Appendix G.2. Event Segmentation	193
Appendix G.3. Audiences Aggregation	194
Appendix H. Visualization	195
10. Annexes	198
Annex A. RDBMS installation	198
Annex A.1. Virtual Machine configuration	198
Annex A.2. Operating System installation	198
Annex A.3. Operating System configuration	200
Annex A.4. Database installation	202
Annex B. Hadoop cluster installation	203
Annex B.1. Virtual Machine configuration.....	203
Annex B.2. Operating System installation	204
Annex B.3. Operating System configuration.....	205
Annex B.4. Node cloning.....	208
Annex B.5. Cluster pre-installation	209
Annex B.6. Cluster installation	210
Annex B.7. Issues found	214

LIST OF FIGURES

Figure 1.1. Dissertation structure	4
Figure 2.1. Visual representation of a Relation, Tuple and Attribute	7
Figure 2.2. SQL's three types of commands	9
Figure 2.3. Data warehouse architecture with data marts	12
Figure 2.4. Star schema diagram	15
Figure 2.5. Snowflake schema diagram	15
Figure 2.6. The DW 2.0 database landscape	16
Figure 2.7. Hadoop 1 stack	19
Figure 2.8. Hadoop 2 stack	19
Figure 2.9. HDFS architecture	21
Figure 2.10. Detailed Hadoop Map-Reduce data flow	23
Figure 2.11. Map-Reduce data flow using Combiners	24
Figure 2.12. Application execution through YARN	25
Figure 2.13. Hadoop 2 + Tez stack	26
Figure 2.14. Logical DAG	27
Figure 2.15. Physical DAG showing the actual execution	28
Figure 2.16. Hive query execution architecture with Tez	30
Figure 2.17. Impala query execution architecture	32
Figure 2.18. IPTV high-level architecture	35
Figure 3.1. The Research Onion	38
Figure 4.1. Current data warehouse architecture	46
Figure 4.2. Proposed data warehouse architecture	47
Figure 4.3. Source file sample	50
Figure 4.4. Data preparation functional model	51
Figure 4.5. Data warehouse functional model	52
Figure 4.6. <i>Channel Tune</i> event transformation DFD	56
Figure 4.7. <i>Program Watched</i> event transformation DFD	56
Figure 4.8. <i>DVR Events</i> transformation DFD	57
Figure 4.9. <i>Event Segmentation</i> example	57
Figure 4.10. <i>Event Segmentation</i> DFD	58
Figure 4.11. <i>Audiences Aggregation</i> DFD	58
Figure 4.12. RDBMS memory configuration	61
Figure 4.13. Staging Area tables physical model in the RDBMS	62
Figure 4.14. Inventory tables physical model in the RDBMS	63

Figure 4.15. Fact tables physical model in the RDBMS	64
Figure 4.16. Aggregation tables physical model in the RDBMS	65
Figure 4.17. Ambari dashboard reporting the cluster overview.....	70
Figure 5.1. <i>Channel Tune</i> transformation benchmarking	75
Figure 5.2. <i>Program Watched</i> transformation benchmarking.....	76
Figure 5.3. <i>DVR Events</i> transformation benchmarking.....	77
Figure 5.4. <i>Event Segmentation</i> benchmarking	78
Figure 5.5. <i>Audiences Aggregation</i> benchmarking	79
Figure 5.6. <i>Channel Tune</i> transformation scalability benchmarking	81
Figure 5.7. <i>Event Segmentation</i> scalability benchmarking	82
Figure 5.8. <i>Audiences Aggregation</i> scalability benchmarking	83
Figure 5.9. Storage usage comparison	84
Figure 5.10. Television <i>Channel Daily Overview</i> dashboard	86
Figure 6.1. Data transformation tasks cumulative execution time	88
Figure 6.2. Scalability effects on the data transformation tasks execution time	90
Figure 6.3. Total storage usage comparison	91
Figure 6.4. Enhanced data warehouse architecture integrating Hadoop.....	93
Figure 9.1. <i>Channel Tune</i> transformation execution plan.....	158
Figure 9.2. <i>Program Watched</i> transformation execution plan.....	158
Figure 9.3. <i>DVR Events</i> transformation execution plan	159
Figure 9.4. <i>Event Segmentation</i> execution plan	160
Figure 9.5. <i>Audiences Aggregation</i> execution plan.....	160
Figure 9.6. <i>Channel Tune</i> transformation DAG	186
Figure 9.7. <i>Program Watched</i> transformation DAG	186
Figure 9.8. <i>DVR Events</i> transformation DAG.....	187
Figure 9.9. <i>Event Segmentation</i> DAG	188
Figure 9.10. <i>Audiences Aggregation</i> DAG	188
Figure 9.11. <i>Intraday Television Share (%)</i> report.....	195
Figure 9.12. <i>Television Channel Reach (%)</i> report.....	196
Figure 9.13. <i>Daily Program Rating (%)</i> report.....	196
Figure 9.14. <i>Weekly VoD Visualizations</i> report.....	197

LIST OF TABLES

Table 1.1. Research objectives	3
Table 2.1. Example of data retrieval with SQL	10
Table 3.1. Research paradigms according to philosophical views	40
Table 3.2. Activity summary according to the Design Science steps	43
Table 3.3. Methodology summary	44
Table 4.1. Design and development activities	45
Table 4.2. Mediaroom event types	50
Table 4.3. Staging Area tables	53
Table 4.4. Inventory tables.....	54
Table 4.5. Support tables	54
Table 4.6. Fact tables.....	55
Table 4.7. Aggregation tables.....	55
Table 4.8. Color scheme for the data volume representation	55
Table 4.9. Hardware specifications of the host computer	59
Table 4.10. Data warehouse environment.....	60
Table 4.11. Tablespace configuration	61
Table 4.12. Implemented transformation processes	66
Table 4.13. Hadoop cluster environment	67
Table 4.14. Hadoop cluster service distribution	68
Table 5.1. Environment configuration for the performance tests.....	73
Table 5.2. Environment configuration for the scalability tests.....	81
Table 6.1. Comparison between RDBMS and Hadoop architectures	92
Table 9.1. <i>Channel Tune</i> event information.....	105
Table 9.2. <i>Box Power</i> event information.....	105
Table 9.3. <i>Trick State</i> event information.....	106
Table 9.4. <i>Program Watched</i> event information	106
Table 9.5. <i>DVR Start Recording</i> event information	107
Table 9.6. <i>DVR Abort Recording</i> event information.....	107
Table 9.7. <i>DVR Playback Recording</i> event information	107
Table 9.8. <i>DVR Schedule Recording</i> event information	108
Table 9.9. <i>DVR Delete Recording</i> event information	108
Table 9.10. <i>DVR Cancel Recording</i> event information	109
Table 9.11. SA_ACTIVITY_EVENTS table information	111
Table 9.12. SA_ACTIVITY_EVENTS information mapping by event.....	112

Table 9.13. SA_ASSET table information.....	113
Table 9.14. SA_CHANNEL_MAP table information	113
Table 9.15. SA_GROUP table information.....	113
Table 9.16. SA_PROGRAM table information	113
Table 9.17. SA_SERVICE table information	114
Table 9.18. SA_SERVICE_COLLECTION table information	114
Table 9.19. SA_SERVICE_COLLECTION_MAP table information	114
Table 9.20. SA_STB table information.....	114
Table 9.21. SA_SUBSCRIBER_GROUP_MAP table information.....	115
Table 9.22. SA_TV_CHANNEL table information.....	115
Table 9.23. SS_ASSET table information	115
Table 9.24. SS_CHANNEL_MAP table information	116
Table 9.25. SS_GROUP table information	116
Table 9.26. SS_MAP_CHANNEL_MAP_SERVICE table information	116
Table 9.27. SS_MAP_STB_CHANNEL_MAP table information.....	116
Table 9.28. SS_PROGRAM table information.....	116
Table 9.29. SS_SERVICE table information.....	117
Table 9.30. SS_SERVICE_COLLECTION table information	117
Table 9.31. SS_SERVICE_COLLECTION_MAP table information.....	117
Table 9.32. SS_STB table information	118
Table 9.33. SS_STB_GROUP_MAP table information	118
Table 9.34. SS_SUBSCRIBER_GROUP_MAP table information	118
Table 9.35. SS_TV_CHANNEL table information	118
Table 9.36. LU_DATE table information.....	119
Table 9.37. LU_START_GP table information.....	119
Table 9.38. FACT_ACTIVITY_EVENTS table information	120
Table 9.39. FACT_ACTIVITY_EVENTS information mapping by event	121
Table 9.40. FACT_EVT_SEGMENTED table information.....	121
Table 9.41. AG_LIVE_RATING_DY table information	122
Table 9.42. AG_LIVE_REACH_DY table information	122
Table 9.43. AG_LIVE_SHARE_GP table information.....	122
Table 9.44. Oracle 'write' compression test	123
Table 9.45. Oracle 'read/write' compression test.....	123
Table 9.46. Hive 'write' compression test.....	161
Table 9.47. Hive 'read/write' compression test	161
Table 9.48. <i>Channel Tune</i> execution statistics in the RDBMS.....	189

Table 9.49. <i>Channel Tune</i> execution statistics in Hive	189
Table 9.50. <i>Program Watched</i> execution statistics in the RDBMS	190
Table 9.51. <i>Program Watched</i> execution statistics in Hive	190
Table 9.52. <i>DVR Events</i> transformation execution statistics in the RDBMS	190
Table 9.53. <i>DVR Events</i> transformation execution statistics in Hive	191
Table 9.54. <i>Event Segmentation</i> execution statistics in the RDBMS	191
Table 9.55. <i>Event Segmentation</i> execution statistics in Hive	191
Table 9.56. <i>Audiences Aggregation</i> execution statistics in the RDBMS	192
Table 9.57. <i>Audiences Aggregation</i> execution statistics in Hive.....	192
Table 9.58. <i>Channel Tune</i> execution statistics in Hive (scaled-out).....	193
Table 9.59. <i>Event Segmentation</i> execution statistics in the RDBMS (scaled-up)	193
Table 9.60. <i>Event Segmentation</i> execution statistics in Hive (scaled-out)	194
Table 9.61. <i>Audiences Aggregation</i> execution statistics in the RDBMS (scaled-up)	194
Table 9.62. <i>Audiences Aggregation</i> execution statistics in Hive (scaled-out)	194

LIST OF EQUATIONS

Equation 5.1. Formula to calculate the average execution time of a test scenario	74
--	----

LIST OF ABBREVIATIONS AND ACRONYMS

1:1	One-to-One
1:M	One-to-Many
3NF	Third Normal Form
ACID	Atomicity, Consistency, Isolation, Durability
AL	Activity Logs
ANSI	American National Standards Institute
BSS	Business Support Systems
CBO	Cost Based Optimizer
CPU	Central Processing Unit
CRUD	Create, Read, Update and Delete
DAG	Directed Acyclic Graph
DB	Database
DBMS	Database Management System
DCL	Data Control Language
DDL	Data Definition Language
DFD	Data Flow Diagram
DML	Data Manipulation Language
DVR	Digital Video Recording
DW	Data Warehouse
EDW	Enterprise Data Warehouse
ELT	Extract-Load-Transform or Extraction-Loading-Transformation
EPG	Electronic Program Guide
ETL	Extract-Transform-Load or Extraction-Transformation-Loading
GB	Gigabyte
GPON	Gigabit Passive Optical Network
HDFS	Hadoop Distributed File System

HDP	Hortonworks Data Platform
HiveQL	Hive Query Language
HPL/SQL	Hybrid Procedural SQL
IP	Internet Protocol
IPTV	Internet Protocol Television
IS	Information Systems
JDBC	Java Database Connectivity
JVM	Java Virtual Machine
LLAP	Low Latency Analytical Processing
M:1	Many-to-One
MB	Megabyte
MPP	Massively Parallel Processing
MR	Map-Reduce (in the context of Big Data) and Mediaroom (in the context of IPTV)
NoSQL	Not Only Structured Query Language
ODBC	Open Database Connectivity
ODS	Operational Data Store
OLAP	Online Analytical Processing
OLTP	Online Transaction Processing
ORC	Optimized Row Columnar
OSS	Operational Support Systems
PGA	Program Global Area
PiP	Picture-in-Picture
PL/SQL	Procedural Language/Structured Query Language
RCFile	Record Columnar File
RDBMS	Relational Database Management System
SA	Staging Area
SGA	System Global Area

SQL	Structured Query Language
STB	Set-top Box
SVoD	Subscription Video-on-Demand
TV	Television
VoD	Video-on-Demand
xDSL	Digital Subscriber Lines

1. INTRODUCTION

Big Data, infinite possibilities. The amount of information collected as of 2012 is astounding; around 2.5 Exabytes¹ of data are created every day and this number is doubling every forty months (McAfee & Brynjolfsson, 2012). Like the physical universe, the digital universe is in constant expansion – by 2020 the amount of generated data annually will reach the 44 Zettabytes², and by then we will have as many digital bits as stars in the universe (Dell EMC, 2014). Nowadays technologies under the umbrella of Big Data contribute decisively to the Analytics world (Henry & Venkatraman, 2015) and the availability of huge amounts of data opened the possibility for a myriad of different kinds of analyses that ultimately feed and enable decision support systems (Ziora, 2015). The ability to process these huge amounts of data, one of the key features of Big Data (Jin, Wah, Cheng, & Wang, 2015), is then of great interest to organizations as they acknowledge the benefits that can be extracted from Big Data Analytics (Kacfeh Emani, Cullot, & Nicolle, 2015). Understanding then the importance of Big Data and its contribution to Analytics can be viewed under the simple concept that more is just better, since in data science having more data outperforms having better models (Lycett, 2013).

One source of large amounts of data is the Ericsson Mediaroom, a video platform that delivers Internet Protocol Television (IPTV) services to customers at their homes, like watching Live television or Video-on-Demand (Ericsson Mediaroom, 2016). Underneath this platform sits a Relational Database Management System (RDBMS) where millions of records are stored every day. These records reflect a variety of behaviors that can be performed by the television users at their homes, like changing a channel or a program. Periodically these events are sent to a centralized database and stored there. It is possible then to access them, in this centralized database, but for a limited time-window since the data is refreshed periodically due to volume constraints (Architecture of Microsoft Mediaroom, 2008). Therefore, if we want to store this data for future analyses, we need to extract it from this repository and store it in another location. From the extraction onwards, our research focuses on optimizing the processes surrounding the transformation of this raw data into valuable and actionable information. A comparative study is performed with the purpose of assessing the benefits of incorporating Big Data technologies in traditional data warehouse architectures, typically supported by an RDBMS. To achieve this, the required transformation processes are implemented in both the RDBMS and Hadoop, a software framework for storing and processing large data sets in a distributed environment of commodity hardware clusters (White, 2015).

This research explores the opportunities and challenges provided by the Big Data technological landscape and solves a specific problem that could not be previously solved by traditional relational databases due to the amount of information that needs to be processed.

1.1. BACKGROUND AND PROBLEM IDENTIFICATION

In traditional systems, when more processing capabilities are required, we are forced to expand their processing power by adding more and better resources, namely processors, memory or storage. This approach, known as vertical scalability, has associated high costs and it is constrained by the

¹ One Exabyte is the equivalent of one billion Gigabytes.

² One Zettabyte is the equivalent of one trillion Gigabytes.

architectural design that cannot evolve beyond the finite capabilities of one single node, the server (J. A. Lopez, 2012). In the Big Data world scalability is horizontal – instead of growing the capabilities of the individual servers, the Big Data infrastructure grows by simply adding more nodes to the cluster, the set of computers that work together in a distributed system (Ghemawat, Gobioff, & Leung, 2003). This scalability, when compared to the vertical scalability, offers infinite growing potential while the costs remain linear (Marz & Warren, 2015). Nowadays, due to the amount and speed of information generated from a multiplicity of sources, traditional Data Warehousing tools for data extraction, transformation and loading (ETL) are, in many cases, at the limit of their capabilities (Marz & Warren, 2015). Under these circumstances, the aim of this study is to explore and assess the value of Big Data technologies in the transformation of data, with the purpose of integrating them in traditional Data Warehousing architectures. The goal is not to replace data warehouses by Big Data infrastructures, but instead to put both worlds working together by harnessing the best features of each of them.

As the amount and types of available data have grown in the past years and will continue to grow, there is little doubt that the Big Data paradigm is here to stay (Abbasi, Sarker, & Chiang, 2016). The technologies that support the Big Data problems are relatively new, but their application, alongside traditional legacy Data Warehousing systems, presents itself as an interesting evolution opportunity. With both technologies working together, in a hybrid approach, we can offer the best of both worlds and apply them to warehousing architectures (Dijcks & Gubar, 2014; Russom, 2014).

Adoption of Big Data technology is a hot topic nowadays and the potential benefits are significant but, due to its young age, there are many challenges that need to be carefully addressed (Jagadish et al., 2014). Using Big Data as a transformation tool can be a solid first step in moving towards the world of infinite data. Hadoop's ecosystem has several emerging data warehouse-style technologies that can be used to solve the problems that traditional technologies cannot overcome, when the volume and variety of data steps up (Kromer, 2014). When faced with huge amounts of data, how can traditional data warehouses evolve and maintain their value? This question poses the central topic on which this study is focused. In the end, managers need their questions answered and what is changing is the amount of information that is being used to support these answers (McAfee & Brynjolfsson, 2012).

1.2. STUDY OBJECTIVES

The main goal of this study is to assess and validate the feasibility of Hadoop as a data transformation tool that can be integrated as part of a traditional data warehouse. In other terms, our driving research question relates to the validation of the hypothesis stating that Hadoop can be used to augment traditional data warehouse architectures. To achieve this goal, Hadoop is used to calculate, from television viewing events, relevant television audience measurements like the *Reach*, number of individuals of the total population who viewed a given channel at any time across its time interval; *Share*, percentage of viewers of a given channel at a given time; and *Rating*, average population who viewed a program across its broadcasting time (Mytton, Diem, & Dam, 2016). These metrics are, ultimately, stored in a traditional data warehouse that corresponds to the single version of the truth in serving information to the decision support systems (Krishnan, 2013).

To reach the final goal of this research, we have established a roadmap, with several sequential steps, to guide us through in a measurable and successful way. In order to explore the data, with the use of

Big Data technologies, there is the necessity of designing and creating a Big Data Hadoop cluster. This lays the foundation for the study to implement the required processes and evaluate their results and performance.

It is also an objective of this study to assess the horizontal scalability potential that is offered by Hadoop clusters. The design of a solution for a problem should remain valid no matter the volume of data we intend to process. This is one of the greatest advantages of using Big Data technologies and in particular the Hadoop Distributed File System (HDFS) (White, 2015). When compared with traditional ETL technologies, Big Data technologies are able to scale seamless horizontally and adapt to big volumes of data (Kromer, 2014).

Table 1.1 briefly describes the six objectives that help to methodically reach the main goal of this research.

Objective	
O.1	Investigate Hadoop's state of the art and determine which solution is more appropriate for the problem at hand
O.2	Install a Hadoop cluster to serve as the foundation for the study
O.3	Make use of the selected Hadoop solution to transform Mediaroom's raw data into meaningful television audience measurements, the <i>Rating, Reach</i> and <i>Share</i>
O.4	Measure and compare the performance of the implemented transformation processes in Hadoop against their performance in an RDBMS
O.5	Measure the processing scalability offered by the Big Data infrastructure and its corresponding performance improvements
O.6	Make the calculated audience measurements available through a visualization layer

Table 1.1. Research objectives

1.3. STUDY RELEVANCE AND IMPORTANCE

Big Data made its first steps in the beginning of 2000 and it is still a relatively new phenomenon conquering its space in the corporate world. In recent years, companies invested millions of dollars in creating data warehouses to serve their decision support systems and further investments towards Big Data are still seen by managers with some suspicion since they are convinced that their companies aren't fully prepared for the leap (Barton & Court, 2012). It is important to understand that even though the big step forward given by Big Data is technological, its motivations are driven by the need to find solutions for the problems that became too complex and too expensive to be solved by the traditional technologies and architectures (K. Lopez & D'Antoni, 2014). The landscape of Big Data is already vast and offers many different solutions to face the challenges of the complexity and volume of the available data. With this in mind, a good approach is to start small. A small Big Data initiative is a good starting point from where it will be possible to assess its value and plan for the next steps (Franks, 2012). Therefore, the integration of Hadoop as a transformation tool, converting the raw data into actionable information, in the context of the traditional Data Warehousing ETL layer, can solve the problems created by the volume increase of data and also move the organizations' data strategies towards the world of Big Data and its possibilities (J. A. Lopez, 2012).

Besides exploring the general applicability of Hadoop as a transformation tool, and with that extend organizations' current Data Warehousing capabilities, this study creates the procedures required to calculate television audience metrics from IPTV infrastructures like Ericsson's Mediaroom. Nowadays, to collect audience measures we do not need to resort to population sampling anymore. Television service providers have all the data they need, what is missing are simply the right tools to convert this data into actionable knowledge. Thus, the importance of this study can be measured in two ways. On the one hand, it demonstrates how Hadoop can expand current data warehouse capabilities, and on the other hand, for television service providers, the study offers a tangible and scalable way of converting their current raw data into useful audience measurements.

1.4. DOCUMENT STRUCTURE

This dissertation follows the sequence of steps depicted in the diagram below.

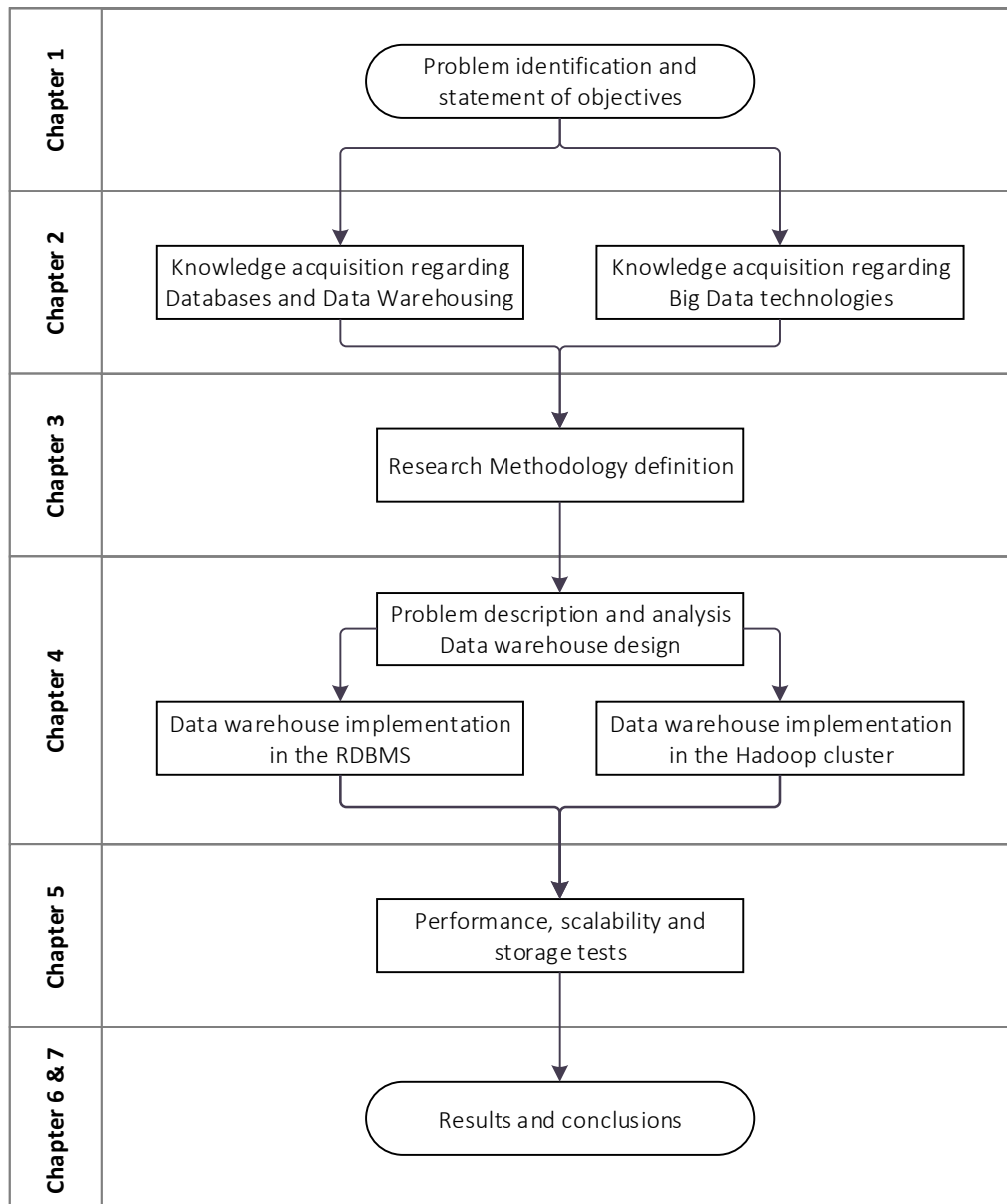


Figure 1.1. Dissertation structure

2. THEORETICAL FRAMEWORK

2.1. INTRODUCTION

The theoretical framework that supports this study crosses a wide-range of theories and techniques. Invariably, there is the need to go back to the origins of relational databases and from there expand the knowledge towards the focal point of the work, the current paradigms around the explosion of information under the umbrella of Big Data. There are many theories and emerging technologies that needed to be analyzed before we could start the implementation phase of this study.

Since the first ideas for the relational databases, proposed by Codd in 1970, the Relational Database Management Systems (RDBMS) have been the norm. With Codd's ideas as a foundation, Online Transaction Processing (OLTP) systems proliferated within organizations; their features multiplied and their applicability allowed for a big dissemination and adoption in a wide range of Information Systems (IS). Relational databases, managed in OLTP systems, became the core of information in organizations, no matter their business purposes (Krishnan, 2013).

The myriad of applications that OLTP systems were supporting created a big divide in information inside organizations. The information was there, but the several systems did not communicate among themselves. With the purpose of creating a more systemic view of the organizations' activities, the first concepts of Data Warehousing emerged in the late 1970s and early 1980s (Krishnan, 2013). The need for transforming data from many sources into useful insights paved the road for the importance of Business Intelligence and, for example, Enterprise Data Warehouses (EDW). Ralph Kimball is one the central figures in the world of data warehouses and in particular in the definition of the dimensional modeling, a variation of the traditional relational model, and the basis of data warehouses' design (Kimball & Ross, 2013).

Within the world of data warehouses and Business Intelligence, there are several techniques for the purpose of providing valuable insights that can be explored through the transformation of data into, ultimately, actionable knowledge. Making use of the dimensional modeling defined by Kimball, the Online Analytical Processing (OLAP) is an extremely useful approach for delivering fast answers from different perspectives (Codd, Codd, & Salley, 1993). Business Intelligence and Analytics have relied, for many years, on the dimensional models and online analytical processing tools that enable the exploration of business information. The standard approaches, that are associated with data warehouse systems, are still very much alive and proof of that is that 80–90% of the deliverables in Business Intelligence initiatives are supported by OLAP built upon an RDBMS-based data warehouse (Russom, 2014).

The explosion of the amount of generated data, and the quest for the most up-to-date information to base decisions upon, created challenges in the traditional Information Systems. Internet giants like Google and Facebook had to change their IS architectures. In 2004, the information regarding the Map-Reduce paradigm was publicly released (Dean & Ghemawat, 2004). Map-Reduce is a programming model focused on the parallel processing of large datasets across an infrastructure composed of multiple computers. This paradigm is implemented by several frameworks, however the best known is Apache Hadoop (White, 2015). Map-Reduce is among one of the changes in how information is processed but, of course, it is not the only one. A plethora of databases that intended to break the

barriers of Codd's relational model were created and as a result, the NoSQL (Not only SQL³) paradigm gained popularity and momentum. NoSQL options, when compared with the traditional RDBMSs, are very simple in their sophistication levels. RDBMSs evolved through decades and these new approaches, in a technological view, look like a return to the past (Mohan, 2013). Traditional RDBMSs are also evolving to allow horizontal scalability while maintaining the integrity of a dimensional database. The Massively Parallel Processing (MPP) paradigm can be seen as a response, by the traditional RDBMSs, to the vast amounts of data and it represents an important approach in the Big Data world (Stonebraker et al., 2010).

The purpose of data warehouses and Big Data, within organizations, is seen through different eyes by several authors. While data warehouses provide a source of clearly defined and unified information that can then be used by other systems like Business Intelligence tools (Kimball & Ross, 2013), some authors state that purpose of Big Data is to provide cheap solutions to store raw data that hasn't any predefined structure (Boulekrouche, Jabeur, & Alimazighi, 2015). This idea is even emphasized by authors advocating that there is no correlation between data warehouses and Big Data, since the latter is only seen as a technology for storing data (B. Inmon, 2013). Moreover, in the opposite side, some defend that Big Data itself consists of both technologies and architectures (Maria, Florea, Diaconita, & Bologa, 2015). The Data Warehousing Institute strongly believes that Hadoop cannot replace a traditional data warehouse since, for example, enterprise data reporting requirements cannot be satisfied by Hadoop as well as they can be by an RDBMS-based data warehouse. Technologies have completely different levels of maturity, and in the end, the most important aspect is which approach can better suit the specific objectives (Russom, 2014).

Hadoop can be seen as the next step in the development of data warehouses and especially in the Extract-Transform-Load (ETL) phase, even though Hadoop is not an ETL tool (Šubić, Pošćić, & Jakšić, 2015). Combining new technology as an integrator of data in a traditional data warehouse is explored so that its advantages and shortcomings can be assessed in an empirical way that goes beyond the theory and the so many contradictory opinions in the world of data science.

2.2. RELATIONAL DATABASES

When we speak about relational databases it is mandatory to go back to 1969-1970 and more specifically to the seminal publications "Derivability, Redundancy and Consistency of Relations Stored in Large Data Banks" (Codd, 1969) and "A Relational Model of Data for Large Shared Data Banks" (Codd, 1970) by Edgar Frank Codd. Codd's second publication is the single most important event in the history of databases and its proposed relational model is still a part of the vast majority of databases (Date, 2003). The relational model appears as a response to the challenges of managing a growing amount of data. It is primarily focused on the independence between data and applications, and with the problems of inconsistency typically associated to data redundancy.

Within Codd's proposal, for the original relational model, we can discern three major components – *structure*, *integrity* and *manipulation* (Date, 2015). As structure features, firstly we have the *relations* that are commonly referred as tables. These relations contain *tuples* that can be interpreted as the

³ SQL stands for Structured Query Language and it is approached in section 2.3.

rows of a table, and finally, a tuple is constituted by a set of *attributes* that instantiate values from the valid *domains* (also referred as types).

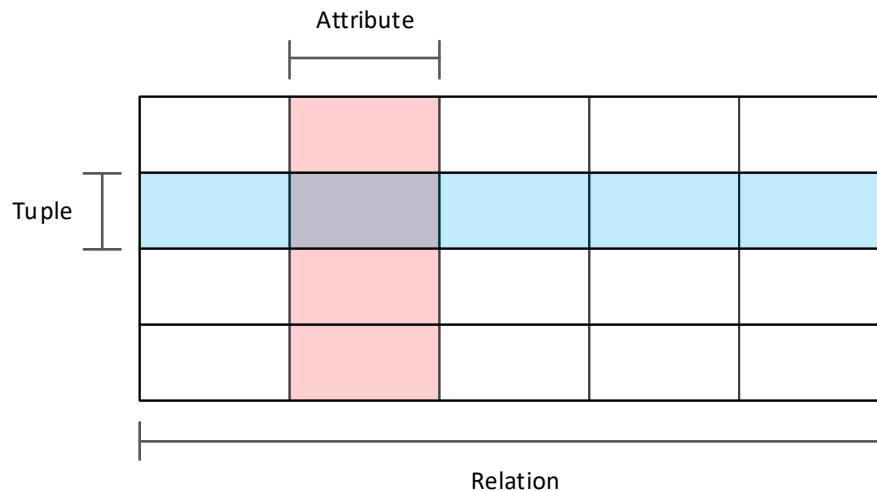


Figure 2.1. Visual representation of a Relation, Tuple and Attribute

In Figure 2.1 we have a visual representation of a relation with a set of tuples containing each five attributes. This relation can also be expressed as a table containing multiple rows, each with five columns. In the depicted example, we are considering a five-ary relation since the number of attributes (or columns) is used to express the arity of the relation. Attributes inside relations contain actual values within their corresponding domain. As an example, an attribute expressing European countries could only contain values belonging to the conceptual pool of European countries.

Still, regarding the structure of the relational model, we have to consider several kinds of keys. Relations need to have at least one *candidate key* capable of expressing the uniqueness of a tuple. These keys can be composed of one or more attributes. What is required is that the relation has a unique key value for each distinct tuple. From the candidate keys, we can elect one to be the *primary key* and thus being subject to special treatment. Finally, we have the *foreign key* that is defined by one or more attributes in a given relation *R2* that must also exist as a key *K* in some other relation *R1*.

When we approach the aspects of the model structure, that emphasize the importance of data consistency, it is important to mention data normalization and the normal forms. The goal of data normalization is to reduce or even eliminate data redundancy. That is expressed in the model definition by splitting relations, with redundant information, into two or more relations. Even though relational theory defines several normal forms, the one most commonly used in relational models is the *Third Normal Form* (3NF) since it covers most of the anomalies related to the update of redundant data (Sumathi & Esakkirajan, 2007). The 3NF builds upon the two previous normal forms and states the following:

1. All attributes contain only atomic values (*First Normal Form*);
2. Every non-key attribute is fully dependent on the primary key (*Second Normal Form*);
3. Every attribute, not belonging to a candidate key, is non-transitively⁴ dependent on every key.

⁴ A transitive dependency is when an attribute is only functionally dependent on the key indirectly, i.e. through another non-key attribute.

The *integrity* features, from the relational model main components, allow us to enforce rules in the model, with the purpose of assuring consistency in the data and in the relationships that help shape and express the logical concepts into the relational model. Simply put, the original relational model bases its integrity features in two generic constraints, one related to the primary keys and another related to the foreign keys. The *entity integrity rule* states that primary keys should represent uniquely the tuples and for that purpose they cannot contain null values, and the *referential integrity rule* establishes that there cannot be any unmatched foreign key values in the corresponding target candidate key. A more implicit constraint can also be considered, the *attribute integrity* that states that an attribute value must belong to its specified domain.

The *manipulative* features enable us to query and update the data stored in the model. They relate to relational algebra and relational algebra assignment which allow us to assign the value of a given relational algebra calculation to another relation (e.g., $R3 = R1 \text{ INTERSECT } R2$). In relational algebra, we can identify the following original operators:

- *Restrict* – returns a relation filtered by a given expression;
- *Project* – defines the attributes that will be part of the relation;
- *Product* – returns a relation containing all the possible tuples resulting from the combination of two tuples belonging to two different relations. This operator is also known as *cartesian product/join*;
- *Union* – returns a relation containing the tuples that exist in any of two given relations;
- *Intersect* – returns a relation containing the tuples that exist in both the specified relations;
- *Difference* – returns a relation containing the tuples that exist in the first relation but not in the second;
- *Join* – originally named *natural join*; it returns the tuples that are a combination of two tuples from two distinct relations that share the same values in the common attributes.

The relational model itself, as published by Codd, does not establish a formal language per se to implement these operators and enable the described manipulative features. That role was taken afterward by SQL (Structured Query Language).

2.3. STRUCTURED QUERY LANGUAGE

SQL was originally developed by IBM, under the name of Structured English Query Language, with the purpose of manipulating and retrieving data stored in Codd's relational model. Its first commercial implementation was released in 1979 by the company that is nowadays Oracle (Oracle Corporation, 2016). Even though SQL's design foundation was Codd's relational model, it deviates in some ways from its original definitions. In SQL, for example, we can apply order to the data retrieval, and tables are viewed as a list of rows instead of a set of tuples. On this spectrum, we can find criticism arguing that SQL should be replaced by a language strictly based on the original relational theory (Darwen & Date, 1995).

SQL is the standard language used by RDBMSs to access and manipulate data in relational databases. It is a nonprocedural language and, therefore, the user only needs to state which data is to be retrieved, without having to specify how the data should be obtained (Sumathi & Esakkirajan, 2007).

RDBMSs commonly also include procedural languages that extend SQL features like Oracle's PL/SQL (Procedural Language/Structured Query Language) or Microsoft's Transact-SQL. As mentioned, SQL is indeed standard for relational databases, but despite this, many RDBMSs vendors do not follow strictly the standard convention, currently the SQL:2011, and implement their own variations. However, the ANSI⁵ SQL standard is supported in most major RDBMSs, thus making SQL interoperability a great asset that contributed to its adoption.

SQL's vendor independence based on official standards and associated to a high-level, English-like language, are some of the aspects that contributed to SQL's success (Weinberg, Groff, & Oppel, 2010). This complete and common language for all relational databases also assures that the developers' skills remain valid when moving from one vendor to another since all programs written in SQL are portable and require little or no modification for them be moved from one RDBMS to another (Oracle Corporation, 2016).

As stated, SQL enables its users to retrieve and manipulate data stored in relational databases. For that purpose, SQL has a set of predefined commands that can be divided, according to their scope, in three types – the Data Manipulation Language (DML), the Data Definition Language (DDL), and finally the Data Control Language (DCL), that according to ANSI SQL is considered to be a part of the DDL.

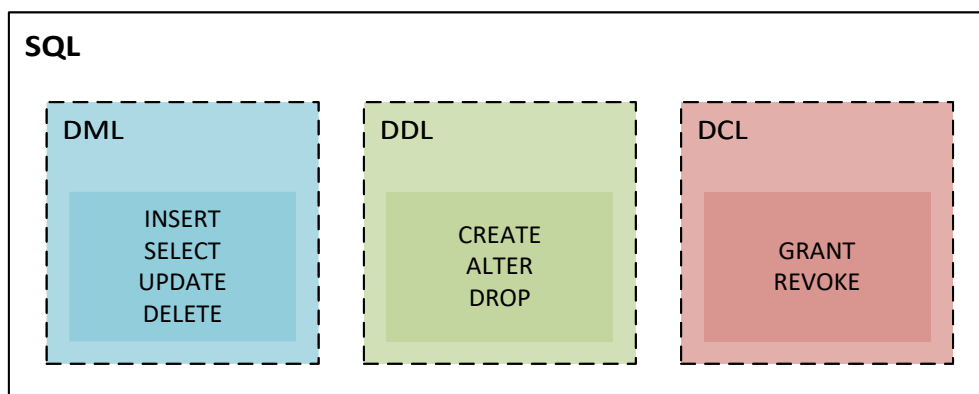


Figure 2.2. SQL's three types of commands

From the figure above we can discern SQL's three types of commands. DML implements the four basic functions of persistent storage – *create*, *read*, *update* and *delete* (CRUD) (Martin, 1983). DDL is used to create, alter or drop objects like tables, views, constraints or indexes. Moreover, the DCL commands allow us to control access to the database objects by granting or revoking permissions to users.

SQL is much more than a query language as its name suggests. Querying data is one of the most important functions performed by SQL but it is not the only one. It enables the users to control many aspects of a Database Management System like:

- *Data retrieval* – through SQL users can query the database and retrieve the desired information. One example of data retrieval statement is presented on Table 2.1;
- *Data manipulation* – applications or users can add new data and update or delete previously stored data;

⁵ American National Standards Institute (ANSI) is a non-profit organization that promotes and facilitates voluntary consensus standards and conformity assessment systems.

- *Data sharing* – SQL is also used in the coordination of concurrent accesses by assuring ACID⁶ properties in the transactions (Haerder & Reuter, 1983);
- *Data definition* – SQL lets the users define the structure and the relationships of the model responsible for storing the data;
- *Data integrity* – to avoid data inconsistency SQL allows for the definition of constraints that assure data integrity;
- *Access control* – SQL can be used to protect data against unauthorized access. Through the Data Control Language, it is possible to restrict the users' ability to retrieve, add, modify or delete data.

SQL Statement	Operator	Description
SELECT <i>t1.column1, t2.column3</i>	PROJECT	Defining the data retrieval projection
FROM <i>table1 t1</i>	<i>Source</i>	Retrieving data from a given table
JOIN <i>table2 t2</i> ON (<i>t1.column1 = t2.column1</i>)	JOIN	Performing a <i>join</i> with a second table
WHERE <i>t1.column2 >= value</i>	RESTRICT	Restricting the output by a given criteria
ORDER BY <i>t1.column1 ASC</i>	<i>Order</i>	Specifying the result order

Table 2.1. Example of data retrieval with SQL

In Table 2.1 we can see an example of a SQL query that will retrieve data from the data model. In the column 'SQL Statement' the language keywords are highlighted, and in the column 'Operator' we can observe, also highlighted, the original operators from relational algebra. Note the use of aliases when referring to the tables to simplify references to their columns (e.g.: *table1* has the alias *t1*).

SQL databases are extremely powerful as they enable the combination and analysis of data through the simplicity of relational algebra. They can be used to represent a single point in time and a single point in space through their transactional serializability and their clear context isolation, respectively. As a language, SQL allows the developers to easily express their intent and navigate the relational model effectively and efficiently (Helland, 2016).

2.4. RELATIONAL DATABASE MANAGEMENT SYSTEMS

We can define a database as an organized and interrelated collection of data, modeled to represent a specific view of reality, and a Database Management System (DBMS) as a complex system with the purpose of managing databases (Date, 2003). The DBMS essentially manages three aspects: the *database schema* that defines the data structure, the *data* itself and the *database engine* that enables the interface between the users and the data.

The categorization of a DBMS can be done by its underlying implementation model, and therefore we can simply just say that a Relational Database Management System is a DBMS that manages a relational database and, in most cases, uses SQL. This is a simplistic view since to correctly classify a DBMS as relational it must adhere to "Codd's 12 rules" (Codd, 1985).

⁶ ACID (Atomicity, Consistency, Isolation, Durability)

One of the objectives of a DBMS is *data availability* and, to that end, it acts as an interface between the users and the data by translating the physical aspects, involved in storing and organizing the data, into a logical view that can be easily accessed by users and applications. The DBMS is also responsible for the correctness of the data it stores, and thus *data integrity* is a fundamental objective present in these management systems. When providing data to the users, a DBMS needs to have a set of functionalities with the purpose of assuring that only authorized users can retrieve and manipulate the data they are intended to and this takes us to another objective of a DBMS, *data security*. Finally, a DBMS also provides an abstraction layer of how data is stored inside the database, through the use of complex internal structures, specifically concerned with storage efficiency. This is *data independence* and allows for the users to store, manipulate and retrieve data efficiently (Sumathi & Esakkirajan, 2007).

Database management systems provide several benefits that empower users in their data management activities. Users understand the logical view of data without having to concern with the complex physical mechanisms that work to assure that the correct data is available in the most efficient way. DBMSs provide a centralized data management entity where all the data and related files are integrated in one single system. With this, data redundancy is minimized while, at the same time, its consistency and integrity can be more effectively assured. DBMSs also play an important part in application architecture since they enable independence between applications and data.

When we speak about Database Management Systems we are abstracting ourselves from the underlying model present in their database engine, but, in most cases, we are implying that we are dealing with the relational model and, therefore, the DBMS is an RDBMS. This is true because even though there are many approaches to the data model, it is the relational model that is effectively the most important one from both theoretical and economic perspectives (Date, 2003).

2.5. DATA WAREHOUSING

The benefits provided by DBMSs made data a more accessible asset and enabled the creation of effective applications and systems throughout organizations with the purpose of supporting their specific processes. These systems were initially individual entities that managed their information and this led to an uncontrolled proliferation of data. Organizations had in fact data about their activities, but it was hard to find it and even harder to assert if it was correct. The existence of a multitude of departmental truths, solely based on isolated views, made the task of finding the single organizational version of the truth very difficult (W. H. Inmon, 2005). This need for an organization-wide single version of the truth, which could be easily used by the decision support systems, triggered a paradigm shift in information architecture that consequently gave origin to the concept of Data Warehousing (W. H. Inmon, Strauss, & Neushloss, 2008).

As a unifying repository, containing nonvolatile data, that is both granular and integrated, a data warehouse (DW) is a basis for information processing that aims to support management decisions. Besides integrating detailed information from various systems, the DW is also time variant since it can store historical data for several years. This combination of factors enables multiple subject-oriented views that, even though are based on the same single truth, can have different levels of detail and different temporal scopes (W. H. Inmon, 2005). Data Warehousing is the process of capturing data

from transactional operational systems, transforming it into meaningful information so that it can be easily accessed by the users in order to promote data analysis and enable fact-based business decisions (Kimball & Caserta, 2004).

Data warehouses and their architectures vary according to the specific realities and requirements of each organization. We can find multiple architectures and also different design approaches when it comes to building a DW (Sumathi & Esakkirajan, 2007). In Figure 2.3 we present a high-level architecture for a data warehouse designed with the *top-down* approach. This approach consists of firstly creating the data warehouse itself, with an enterprise-wide vision, and then expanding it by adding data marts with more subject-oriented views. The top-down approach provides consistent dimensional views across all data marts since they are generated from the data warehouse and not from the operational systems. This centralized approach is also robust when it comes to business changes since the implementation of data marts only depends on the data warehouse. However, designing a full data warehouse requires considerably more effort, and thus costs, when we compare it to the implementation of single data mart from an operational system as it is proposed by the *bottom-up* approach. With the bottom-up approach it is possible to deliver faster results since implementing single data marts, as the requirements evolve, requires lower initial investments than creating the full enterprise-wide data warehouse. The data warehouse is then built from the information in the data marts (Imhoff, Galemme, & Geiger, 2003; Sen & Sinha, 2005). Both approaches have their advantages, and it is possible to use them together in a *hybrid* approach that combines the development speed and the user-orientation of the bottom-up approach with the enterprise-wide integration enforced by the top-down approach (Kimball & Ross, 2013).

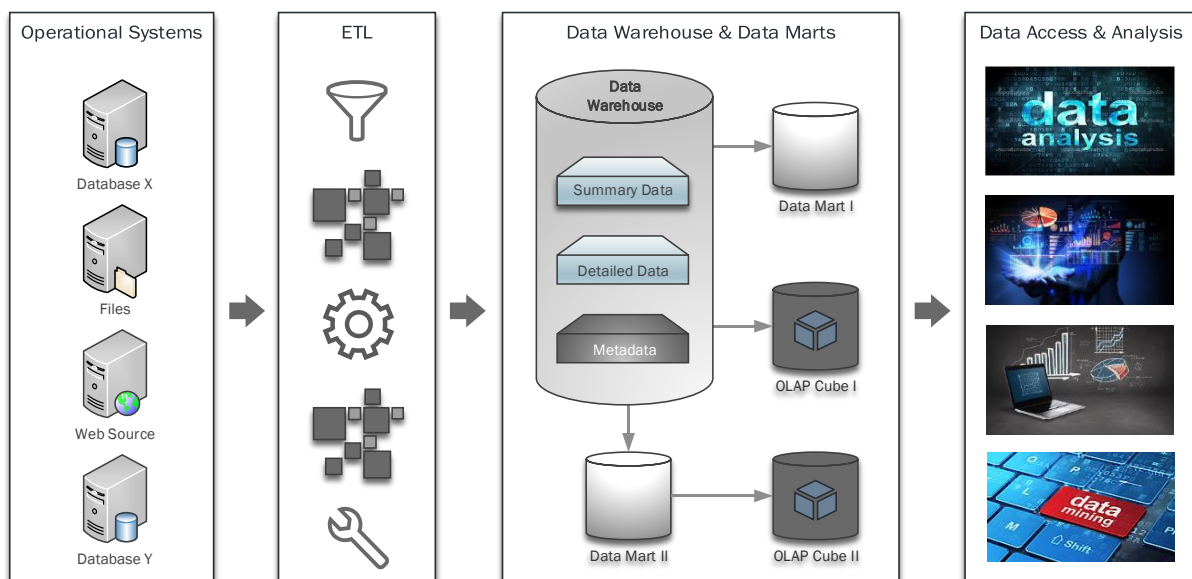


Figure 2.3. Data warehouse architecture with data marts

As mentioned, we can have various data warehouse architectures with the purpose of better suiting the specific informational needs and constraints of a given situation. In a broad view, we can define the architecture of a data warehouse, and the related Business Intelligence applications, as a set of tiers. Not all the architectures have the same tiers, and some tiers can be more or less merged, this, of course depending on the specific scenario. In Figure 2.3 we depicted the data warehouse architecture as being composed by three tiers – the *Extraction-Transformation-Loading*, the *Data Warehouse* itself that, in this case, is expanded by data marts and also multidimensional aggregated objects, and finally

the *Data Access and Analysis* tier that is composed by a set of tools that make use of the data, ranging from simple descriptive analytics, like reporting, to predictive analytics that can, for example, include data mining. We could also easily split the data warehouse and data marts tier into two different tiers, one related only to data storage that would include the data warehouse and the data marts and another tier specifically dedicated to analytics that would contain for example the OLAP cubes. Nevertheless, in all Data Warehousing architectures, we can identify a layer that represents the data sources, presented in Figure 2.3 as the *Operational Systems*. The data sources themselves are not part of the data warehouse architecture per se, even though they are critical to its design, but they rather represent the source of all data that can originate from any system throughout the organization and even from external entities. Therefore, we have considered, as the first tier of the data warehouse architecture, the ETL.

2.5.1. ETL/ELT

The Extraction-Transformation-Loading layer is the foundation of a data warehouse. It is responsible for extracting data from different operational systems and combining it in a way that makes possible to use it together, while at the same time assuring its consistency and quality. Moreover, the role of the ETL is to make this transformed data available for the applications associated to data analysis and decision making. The ETL tier is hidden from the users but its processes, within the DW architecture, are the ones that require more resources for their implementation and maintenance (Kimball & Caserta, 2004).

ETL is far more than just getting data from the sources and deliver it to the users. It can contain complex processes with the purpose of cleaning and conforming heterogeneous sources into a single and unified enterprise-wide view of the captured systems. This flow of data can be achieved by different ETL architectures that can be put into place to better fit the transformation requirements. For example, it is common to find a *Staging Area* within the ETL tier. This area is where the extracted data is placed and subsequently undergoes successive transformations before it can be loaded into the data warehouse. The need for a Staging Area is more or less related to the data quality in the sources and the complexity around their transformation and combination (Malinowski & Zimányi, 2007).

Some early DW architectures also contained an Operational Data Store (ODS) that could be seen as an extension of the ETL layer. The ODS is a hybrid construct that seats between the operational and the decision-support systems. Its purpose is to provide low-latency reporting capabilities that could not be obtained from the DW due to its slower refresh periodicity. Nowadays the use of dedicated ODS is not very frequent and their role was absorbed by the DW itself (Kimball & Caserta, 2004).

An alternative to the ETL paradigm is to directly load the extracted data and only after transform it. By doing so we are implementing an Extract-Load-Transform (ELT) workflow. This approach can be dangerous if we are tempted to disregard completely the transformation processes since the data is already loaded and available in the data warehouse. This would greatly diminish the value of the data and ultimately the value of the DW itself (W. H. Inmon, 2005). Despite this danger, the ELT approach can be used to explore various advantages in the DW architectural design. By performing the transformation inside the database, we have the possibility to use the huge amounts of data already in the DW. The implementation of an ELT architecture also gives us extra flexibility since it facilitates

the inclusion of new data sources and it is more adaptable to business requirements changes. The data is loaded in its raw format, and, therefore, multiple transformations can be applied as the requirements change. ELT addresses some of the inflexibility of ETL when it comes to environmental changes as it brings data closer to the users and speeds up the implementation process (Marín-Ortega, Dmitriyev, Abilov, & Gómez, 2014).

In recent years, the diversification of DW workloads is leading to distributed architectures where, for example, the ETL processes are being offloaded from expensive dedicated platforms to cheaper solutions like Hadoop (Clegg, 2015). Hadoop can be seen as the next step in the design and implementation of data warehouses, with special emphasis in the ETL layer (Šubić et al., 2015). We approach this hybrid architecture that brings Big Data technologies into the data warehouse ecosystem in section 2.8.

2.5.2. Dimensional modeling

When it comes to designing data warehouses, we cannot escape the discussion around Inmon's top-down approach, supported by operational data in the third normal form, and Kimball's bottom-up approach supported by data marts implemented with dimensional models (Breslin, 2004). Both approaches have their differences, as presented previously, but they can be combined in the implementation of data warehouses so that we can benefit from the merits found in each one (W. H. Inmon et al., 2008).

Kimball's dimensional modeling, published in the first edition of "The Data Warehouse Toolkit", became the leading design technique to implement data warehouse models (Kimball & Ross, 2013). The dimensional model is especially oriented to the delivery of data for analysis with emphasis in performance. We can qualify its main benefits, when compared to entity-relationship models, as being a model that facilitates *understandability* and enables *performance* by using a less normalized data model. Data normalization, like the third normal form, is critical to assure data integrity but it has the negative effect of making more difficult to interpret the models and the information they support and, at the same time, it hinders data access performance. Redundancy harms integrity, but it helps performance and understandability. Another benefit, also related to the simplicity of the model, is the *extensibility* that it adds to the design. Due to the simple structure of the model, it is a lot easier to adapt it to new requirements. Adding new concepts to a normalized model requires a great deal of more effort to assure its full consistency, while in a dimensional model it can be achieved by just adding more rows to represent a fact or more columns to identify a dimension (Kimball & Ross, 2013). Extensibility is then easier to implement but, at the same time, it is limited up to a certain degree due to the dependence of the model regarding the initial requirements (W. H. Inmon et al., 2008).

At its core, the dimensional model design revolves around the definition of two main entities that support the representation of a given subject – the *facts* and the *dimensions*. Facts capture and express measurements pertinent to a specific business process. These measurements are then related to entities that place them in a context, the dimensions. Dimensions are used, for example, to express the time of when the measurement occurred or the object being measured, as well as its pertinent properties. The arrangement of fact and dimension tables, to express a given business process, is used to define the model's design in which the most popular are the *star schema* and the *snowflake schema*.

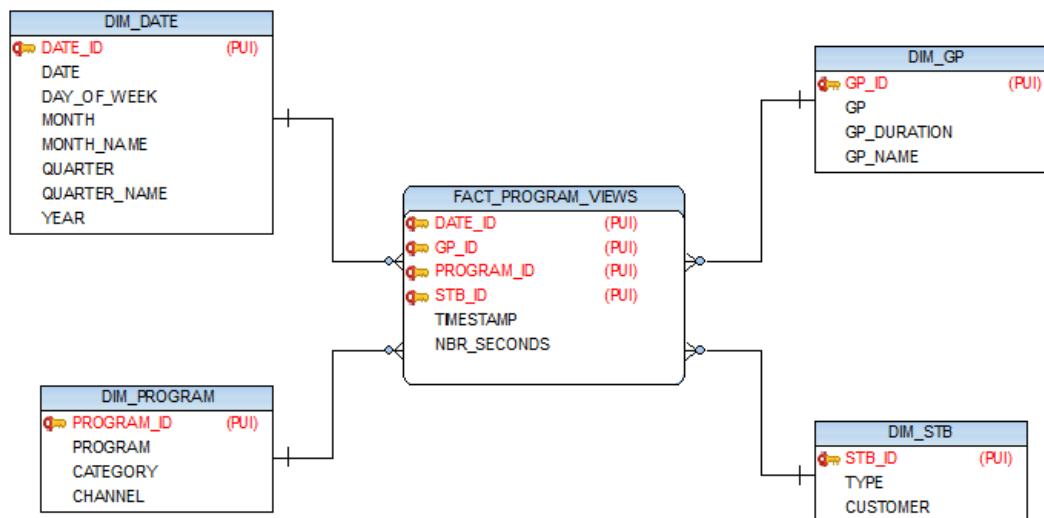


Figure 2.4. Star schema diagram

In the star schema design, shown in Figure 2.4, the model is not in the 3NF, but instead it is denormalized. The process of denormalization consists in reducing the normalization of a given model into a less normalized form, e.g. starting from a model in the 3NF, denormalization can produce a new model in the second or even first normal form. The process of denormalization grants performance benefits to data retrieval since it can eliminate *joins* that otherwise would be necessary to navigate the model but, on the other hand, a denormalized model requires a more complex update process to ensure data consistency because it adds redundancy.

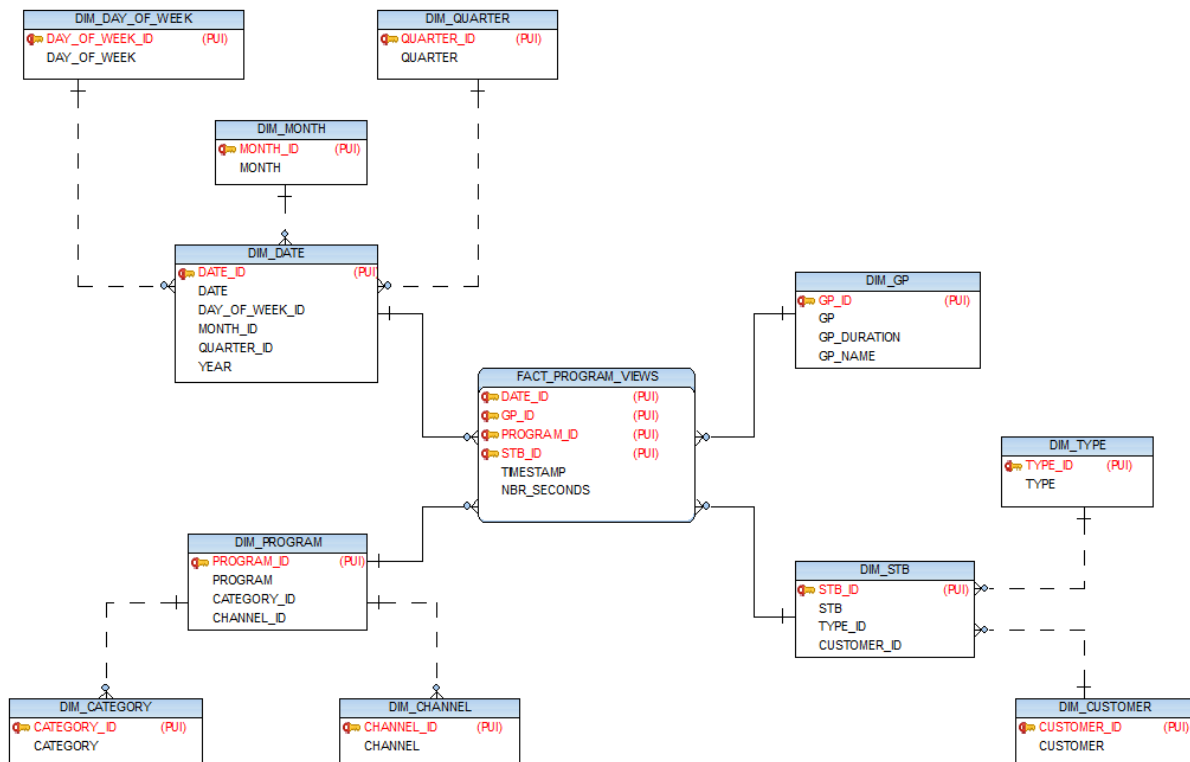


Figure 2.5. Snowflake schema diagram

The snowflake schema, exemplified in Figure 2.5, is a variation of star schema where a fact table is surrounded by multiple dimensions, but the difference is that in the snowflake design, with the

purpose of eliminating redundancy, the dimension tables are normalized, usually in the 3NF, and can be connected to each other through many-to-one relationships.

Choosing between the star schema and the snowflake schema is a balance between complexity, performance, and consistency, where business requirements and the technological infrastructure should also be considered. We can also find other designs like the *starflake*, where we can find both normalized and denormalized dimensions and also the *constellation schema* that consists of multiple fact tables that share the same dimension tables (Malinowski & Zimányi, 2007).

2.5.3. DW 2.0

As businesses and technology evolve, so do IS architectures. The DW 2.0 architecture is the product of evolution amongst established architectures, namely the ones proposed by Inmon and by Kimball. The DW 2.0 paradigm fits together the corporate information factory concept, which advocates the data warehouse as the single version of the truth, with Kimball's architecture centered in data marts. Its focus relates to the basic types of data (structured and unstructured), their supporting structure and how they can relate, in order to establish a central data store capable of satisfying organizations' informational needs and enabling their decision support systems (W. H. Inmon et al., 2008).

The DW 2.0 architecture incorporates several aspects surrounding the diversity of data. Both structured and unstructured data are considered as essential, and the recognition of the lifecycle of data plays a determinant role within this architecture. Also, at the heart of the architecture, metadata is deemed as an essential component containing both technical and business definitions. Metadata in the DW 2.0 represents a cohesive enterprise view capable of capturing and coordinating all sources of metadata distributed across the organization.

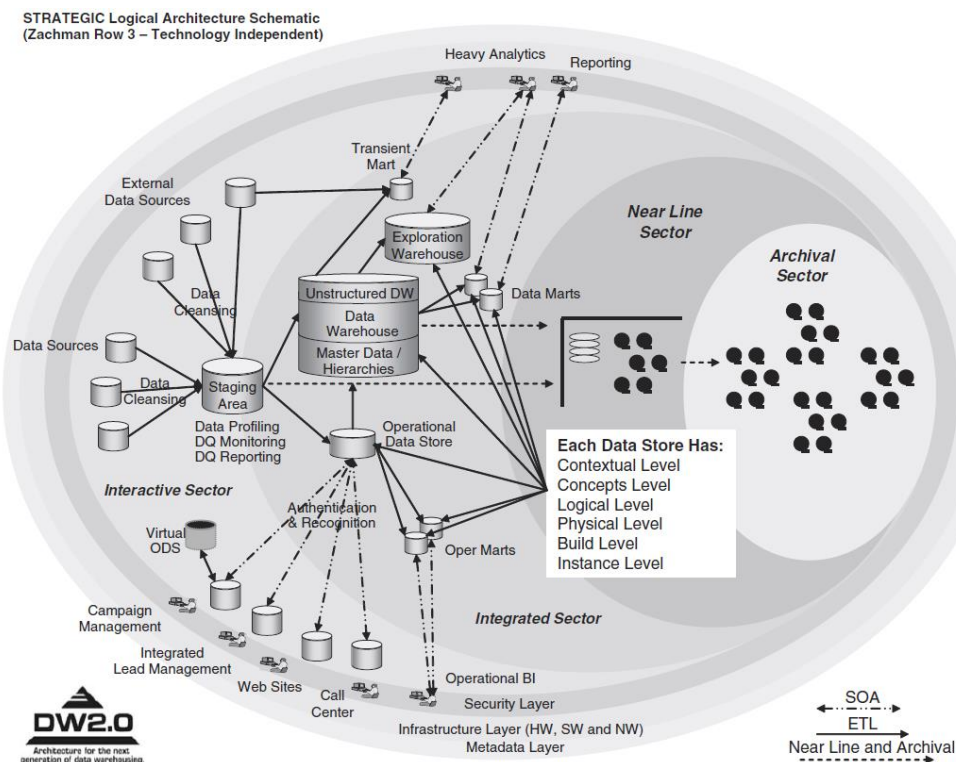


Figure 2.6. The DW 2.0 database landscape (W. H. Inmon et al., 2008)

From the Figure 2.6, and related to the lifecycle of data, we can identify a clear distinction amongst the different sectors of data. From the *Interactive Sector*, that contains the most current data related to the operational systems, to the *Archival Sector* where older data is stored, being the *Integrated Sector* the core of the data warehouse where the data remains until the probability of being accessed declines. In between these two last sectors, we have the *Near Line Sector* that can be seen as an extension of the Integrated Sector and it is optional.

Technology is a facilitator that enhances process efficiency, and it is often driven by challenges triggered by the environment. New types and volumes of data have prompted a constant state of evolution within Data Warehousing architectures and related technologies. The DW 2.0 paradigm is a step forward towards facing the new informational challenges, but in recent years the dimensions of data, namely volume and variety, have been putting to test the capabilities of Data Warehousing architectures. With that in mind, perhaps now, more than a step, we need a leap forward that encompasses the incorporation of technologies from the Big Data spectrum in the architecture of traditional data warehouses and thus the “Modern Hybrid Big Data Warehouse Architectures” (Kromer, 2014). This subject is approached further on in section 2.8 where we identify benefits of incorporating Big Data technologies in the architecture of data warehouses.

2.6. BIG DATA

There are many definitions for Big Data (Dutcher, 2014) but if we just interpret literally its definition we do not find anything new. Big Data means large amounts of data, and it is very easy to assert that this is not a novelty, for example, by looking at its reference in the title of E. F. Codd’s seminal publication from 1970, “A Relational Model for Large Shared Data Banks”. The volume of data has been one of the greatest challenges when it comes to transform data into actionable knowledge. As the volumes of data increase, architectures and technologies evolve, and by doing so, also create new opportunities that quickly are explored to better extract value from data.

There is no clear distinction between what Big Data is and what it is not if we just look at it from the data volume perspective. Within the scope of our study, we can define Big Data as the territory where it is possible to manage large and diverse data sets, with adequate responsiveness, in a way that is no longer achievable through the traditional RDBMSs architectures (Goss & Veeramuthu, 2013). Big Data solutions derive many of their advantages, in the processing of large data sets, from the distributed processing and its almost unlimited horizontal scalability. Traditional RDBMSs rely on a vertical scalability model to increase their capacity and performance whereas Big Data technologies, and their horizontally scalable model, expand or shrink seamless by adding or removing nodes to adapt to the distinct volumes of data (Marz & Warren, 2015).

We already mentioned and emphasized the importance of volume in the characterization of Big Data, but we cannot limit its analysis solely based on this characteristic. To better understand the scope of Big Data it is useful to address its main characteristics commonly referred as the “Vs”. As approached, firstly we have the *Volume* of data, higher than ever and constantly increasing at incredible speeds. This speed at which data is generated and analyzed, to support decision support systems as close to real time as possible, point us to another important characteristic, the *Velocity*. Long gone are the days when data was almost exclusively text. In recent years, data became largely available in a multitude of

formats, like for example sound, image or video. Addressing this *Variety* of semi-structured and unstructured formats is one of the major challenges under the umbrella of Big Data. These first three “Vs” mark the first steps of the Big Data world, but as systems evolved and their applications grew, other concerns were included. Collecting and storing data from various sources, and in very distinct formats, is a complex task that is prone to data quality issues. Issues that are not disregarded but instead are covered by the concerns related to the *Veracity* of data. Collecting and storing large volumes of correct data, in a timely manner, is one part of the equation, the one closer to the data itself, but Big Data accounts for other characteristics relevant also for the consumers of this data. *Variability* addresses the changes of data and its context, especially important for sentiment analysis. *Volatility* accounts for the questions around the validity of data in a temporal perspective, a characteristic particularly important for real-time analysis and in defining the most relevant periods for specific analyses. Big Data opened the door for all kinds of data and empowered analyses that were not possible before and thus it requires new and innovative ways capable of delivering the generated information so that it can be easily understood and leveraged by the end-users. This refers to the usability of data, and it is expressed as the *Visualization* characteristic. Finally, the last “V” used to characterize Big Data is the one directly tied to the goal of data analysis. We use data and its analysis to support the process of decision making so that the extracted insights add *Value* to the organization (Du, 2015; Khan, Uddin, & Gupta, 2014).

Ultimately, we can state that Big Data is a technological enabler for the analysis of huge amounts of data, no matter its structure, with the purpose of creating actionable knowledge and thus value for organizations. Even though Big Data, as a popular phenomenon, is still relatively new, its technological ecosystem is already vast, and it is in a constant state evolution. From the several available tools within Big Data’s universe, we can highlight the two main classes of systems, the NoSQL data stores and Hadoop (Henry & Venkatraman, 2015). NoSQL databases, like MongoDB or Cassandra, are distributed systems that do not rely on the relational model to store their data, while Hadoop represents a broader environment that can also include NoSQL data stores like HBase (Moorthy et al., 2015). The Hadoop ecosystem is approached, in more detail, in the next section.

2.7. HADOOP

Hadoop is a framework designed for the storage and processing of large data sets in a distributed architecture. Hadoop was initially created by Doug Cutting as part of a web-search engine that would be able to search through a high number of web pages (White, 2015). Not long after its first version, Google released one of the most influential papers that helped to shape today’s Hadoop and started the Big Data hype, “The Google File System” (Ghemawat et al., 2003). In this paper, its authors discuss and present a scalable distributed file system that would serve as the foundation for the Hadoop Distributed File System (HDFS) that we will approach in more detail shortly.

Other seminal publication that greatly influenced distributed data processing and particularly Hadoop, was again released by Google, the “MapReduce: Simplified Data Processing on Large Clusters” (Dean & Ghemawat, 2004). Its authors present a simple, yet effective, programming model for the processing of large data sets, the Map-Reduce. Map-Reduce tightly coupled with HDFS defined the shape of Hadoop’s first version.

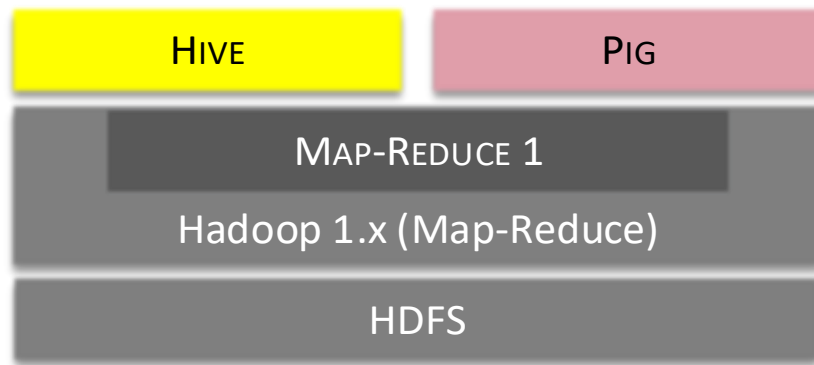


Figure 2.7. Hadoop 1 stack
(Adapted from Saha et al., 2015)

In its first version, Hadoop had a monolithic design where both the processing engine and the resource management were coupled. This meant that all processing activities had to be translated into its programming model and processing engine, the Map-Reduce. In Figure 2.7 we also have the example of two higher level engines, Hive and Pig, that, to process their computations, were required to translate their instructions into Map-Reduce.

The constraining limitations of this architecture were addressed by the design of the second version of Hadoop that clearly separated the resource management from the execution engine. In Hadoop 2, YARN (Yet Another Resource Negotiator) is the layer responsible for the cluster resource management. With this modular approach, the processing engines are able to more easily implement their logic according to the specificities that are part of their purpose and, at the same time, use and share a set of building blocks that are a common part of Hadoop without having to concern with, for example, the resource allocation or management.

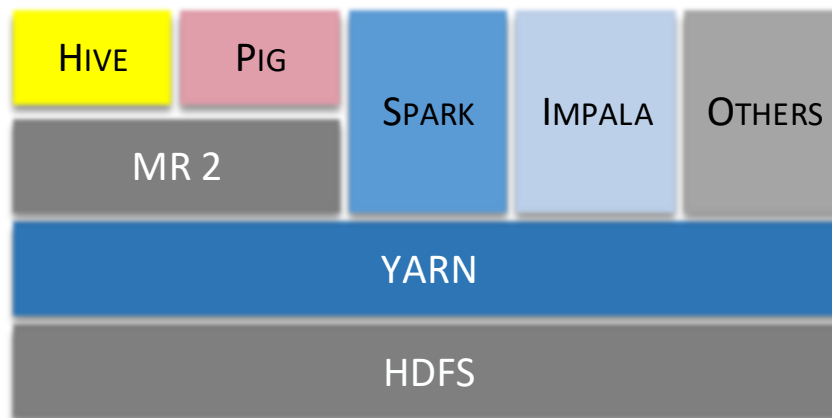


Figure 2.8. Hadoop 2 stack
(Adapted from Saha et al., 2015)

With Hadoop 2, depicted in the figure above, we can observe that some systems, like Impala, a Massively Parallel Processing (MPP) SQL query engine, were able to move away from the Map-Reduce programming model and make use of their own specific implementations to access the distributed storage and resource management layers (Floratou, Minhas, & Ozcan, 2014). In Hadoop 2, once its pillar, the Map-Reduce is now “demoted” and works just like any other application in its ecosystem. This decoupling of applications from the core infrastructure has accelerated innovation, but at the

same time created a less efficient ecosystem, where some common functionalities are being replicated across frameworks (Saha et al., 2015).

Hadoop's differentiating capabilities, like the scalable and the flexible processing of massive volumes of data, no matter its format, associated with its open source design and relatively inexpensive hardware requirements, have revolutionized the world of data management and processing (Grover, Malaska, Seidman, & Shapira, 2014). The enthusiasm created around Hadoop motivated the development of other projects that, in turn, continue to add value to it and diversifying its capabilities. Large scale data analytics, once almost only available to large companies, are nowadays available and critical for most modern organizations (Saha et al., 2015). Despite this, when framed in the world of data management, Hadoop is still a relatively young technology and many organizations are still struggling to understand how it can be used to solve their specific problems (Grover et al., 2014).

2.7.1. HDFS

The Hadoop Distributed File System (HDFS) lays at the bottom of Hadoop's application layer, and it is a distributed file system especially designed to run on commodity hardware. One of the most relevant characteristics of HDFS is that it is designed to work in an environment where the failure of its nodes is the norm and not the exception, as it was established by its most influential publication, "The Google File System" (Ghemawat et al., 2003).

The HDFS architecture is intended to the storage and processing of large data sets, and its design emphasizes the throughput of data, to support batch processing, rather than low latency to enable seamless interactivity with the user data requests (Marz & Warren, 2015). Another important differentiating characteristic of HDFS is its simple coherency model that follows the write-once, read-many schema. HDFS is designed under the assumption that files can only be written once, through create or append, and, therefore, data cannot be updated. This not only facilitates data coherency but also enables high throughput for data streaming (Apache Hadoop, 2013).

HDFS follows a master/slave architecture with a master server, the NameNode, managing the file system namespace and the access requests to the files by the clients, and with multiple DataNodes responsible for managing the data storage attached to them.

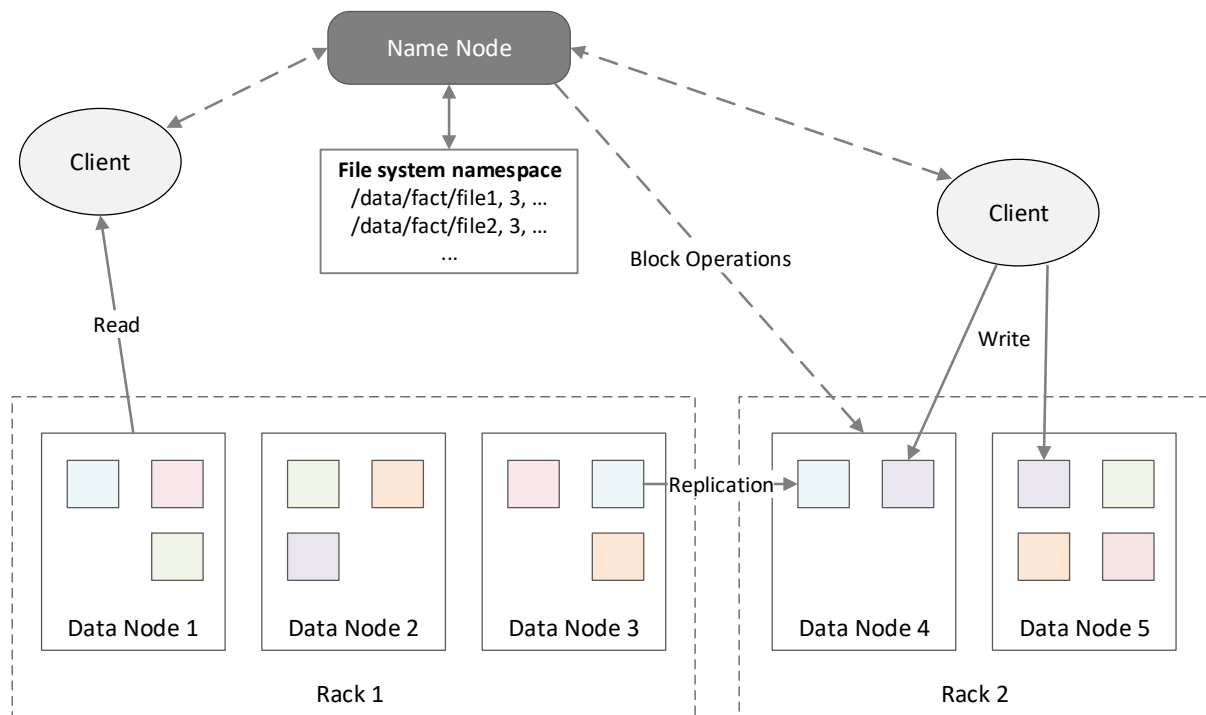


Figure 2.9. HDFS architecture
(Adapted from Apache Hadoop, 2013)

The architecture of HDFS was designed to solve two important aspects in the computation of large data sets – distributed processing and fault tolerance. To enable both these characteristics, the HDFS relies on the ability to break down files in several segments and distribute them across multiple systems. These segments are known as blocks and have a default size of 64MB or 128MB, depending on the version, and a replication factor of three (Krishnan, 2013). Replication has rack awareness, and consequently, the system tries to keep one copy of the original block in the same rack and another in a different rack to assure data availability even in the case of network partitioning⁷.

Figure 2.9 depicts the basics of the HDFS architecture where we have the NameNode controlling the namespace and the accesses to data by the clients. The NameNode role is critical, and in earlier versions of Hadoop its failure would mean that the cluster would stop, but currently, HDFS has redundancy for the NameNode role through Secondary NameNodes. Another important aspect of the HDFS architecture, and showed in the figure above, is how the reads and writes are processed. The NameNode is responsible for storing and managing the metadata regarding the location of the files and blocks but it is not responsible for providing the data itself to the clients. When a client wants to write data, the NameNode manages the allocation of storage space through blocks across the cluster, sends the location where the client should write the data and then it is the client that writes the data itself to the DataNodes. A similar process occurs for reads, the NameNode, through its metadata, knows the location of every file and block in the cluster and tells the clients to fetch the data directly from the designated DataNodes (Krishnan, 2013). The NameNode never acts as a proxy for data between the clients and the DataNodes, but instead, it acts as a manager.

⁷ Network partitioning refers to the situation when communication between nodes is split due to the failure of a network device.

The horizontal scalability of storage in a Hadoop cluster is immense, but ironically it is ultimately constrained by the vertical scalability limits of the NameNode. The NameNode, for performance reasons, keeps all the information about the file system metadata in memory and in the scenario of a cluster with billions of small files, each consuming at least one block, we would require huge amounts of memory on a single machine, the one hosting the NameNode (Shvachko, 2010).

In summary, the HDFS is a file system intended for the distributed batch processing of very large files over commodity hardware and relies on replication, rather than redundancy, to implement high availability and fault tolerance. It is built around the idea of the sequential acquisition of large portions of data, and, therefore, follows the paradigm of write-once, read-many times. Finally, since the priority is to process large sets of data rather than finding individual records, high throughput of data is far more important than low latency.

2.7.2. Map-Reduce

Map-Reduce is a programming model for the processing of large data sets. It was introduced by Google's seminal publication "MapReduce: Simplified Data Processing on Large Clusters" (Dean & Ghemawat, 2004) as a model capable of processing huge amounts of raw data being generated by web related services and applications, in a distributed environment. On the first version of Hadoop, Map-Reduce appears as not only a programming model but also has a resource management framework capable of handling the specificities of distributed parallel processing. With this approach, the users had an abstraction layer that allowed them to focus solely on the data processing without having to deal with the complexity of distributed programming.

Map-Reduce per se is founded on a simple principle defined by two basic functions – the *Map* and the *Reduce* – that are capable of expressing many of the data processing tasks. The *Map* function processes a key-value pair and generates set of intermediate key-value pairs, and the *Reduce* function merges the intermediates values associated with the same key, producing the final result.

The design of this simple approach makes the implementation of parallel processing a lot less complex. Making use of the raw data already split across multiple nodes through a distributed file system like HDFS, Map-Reduce can also split its work through a set of tasks that run on the nodes where the data resides. This important aspect, the data locality, is a fundamental characteristic of Hadoop's design and instructs that Map tasks should run as close as possible to the data, to minimize the overhead generated by the network traffic that occurs if the tasks are not running on the same node as the data (White, 2015).

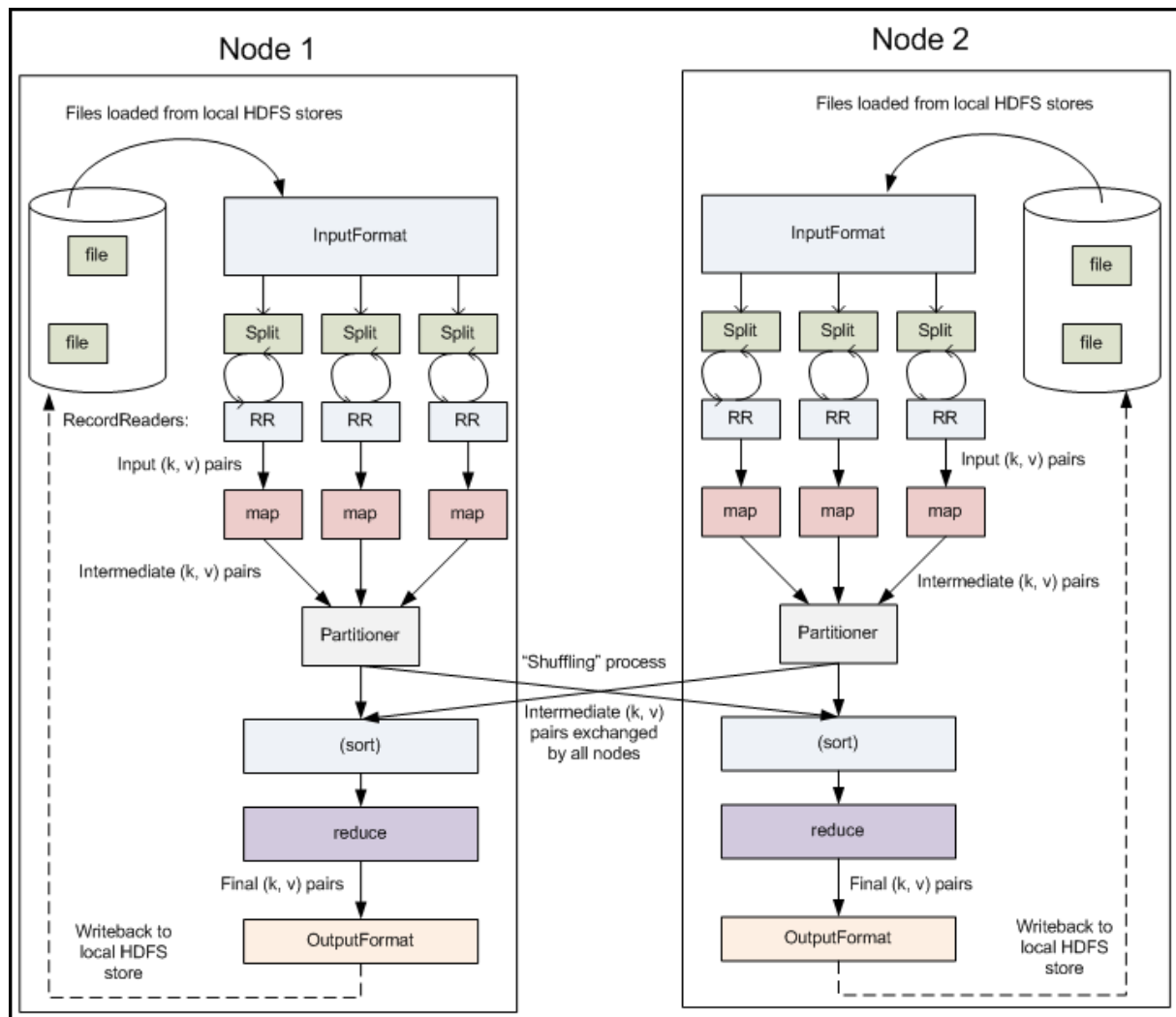


Figure 2.10. Detailed Hadoop Map-Reduce data flow (Yahoo!, n.d.)

From the diagram above, depicting a generic execution of a Map-Reduce job and its associated tasks, we can identify its major participants and understand their roles, namely:

- *Input reader* – it is responsible for reading and splitting the source data from the storage (typically a distributed file system like HDFS). This split data is then passed to the Mappers;
- *Mapper* – the Mapper takes the input data and generates a set key-value pairs according to the implemented rules;
- *Shuffler* – the Shuffler is responsible for transferring the data from the Mappers to the Reducers. In the shuffling phase, the output of the Mappers is firstly partitioned by its key, with the intervention of the Partitioner, so that the values with the same key are sent to the same Reducer. This output can also be sorted to increase the processing speed of the Reducers since that, with a sorted set of key-value pairs, each time a key changes the Reducer can start a new task. The shuffling phase can transfer data between the various nodes in the cluster according to the keys assigned to each Reducer;
- *Reducer* – the Reducer iterates through the values associated with each key and processes them according to the rules defined by the user;
- *Output writer* – the outputs from the Reducers are written to the storage by the Output writers.

Map-Reduce can also incorporate a local *Combiner* that gets the results of each local Mapper and performs pre-defined Reduce-like operations, before sending the data to the Reducers themselves. Through the use of a Combiner, we can optimize the bandwidth and performance of the data flow since the amount of data moving from the Mappers to the Reducers is minimized by the operations performed by the Combiners.

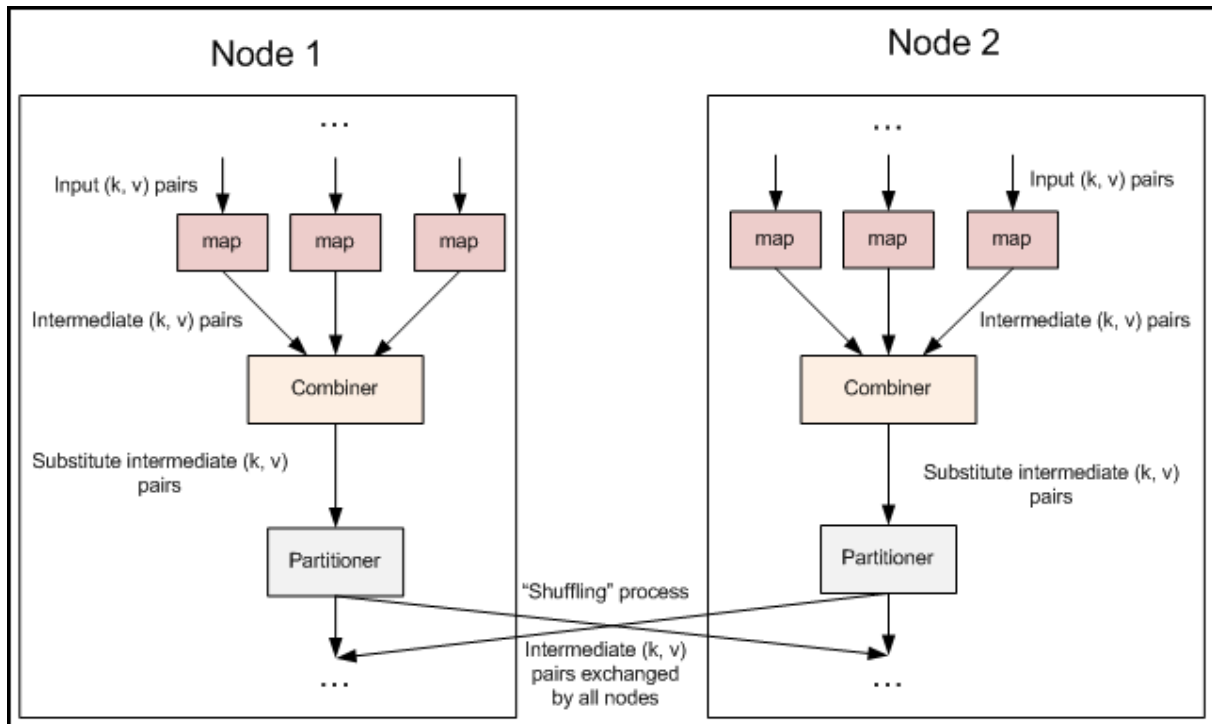


Figure 2.11. Map-Reduce data flow using Combiners (Yahoo!, n.d.)

As it shown in the figure above, it is important to emphasize that even though the Combiners can execute operations that are typical of the Reducers, they are always local, i.e., they can only use the data produced by the Mappers in the node where they are being executed. The Combiner is then a local Reducer-like extension of the Mappers.

Map-Reduce, with its simplistic, but yet effective paradigm, played a pivotal role in re-shaping the world of data science, but its merits are far from being uncontested. Characteristics like the lack of schema definition or features like indexing and storage options are pointed as aspects that hinder Map-Reduce's performance, making it less efficient than parallel databases in several types of analytical processing (Pavlo et al., 2009; Stonebraker et al., 2010).

Map-Reduce, amidst its flaws and limitations, was greatly responsible for the enthusiasm around Big Data and this sentiment, associated with the open-source format of Hadoop's implementation of Map-Reduce, led to many other initiatives that built upon these limitations thus creating a plethora of frameworks and applications that vastly enrich, not only the Hadoop's ecosystem, but also data science in general.

2.7.3. YARN

When we speak about YARN (Yet Another Resource Negotiator) we are referring to the second generation of Hadoop (please refer to Figure 2.8) where the resource management was decoupled from the processing engine, thus originating the creation of YARN, the Hadoop's cluster resource management system. YARN was created not only with the support of Map-Reduce in mind but also with a general purpose design, capable of supporting other distributed computing paradigms (White, 2015).

In YARN we can identify several components that take part in the allocation and management of resources with the purpose of executing data processing activities. We have a global *Resource Manager*, which also includes a scheduler, with the responsibility and authority to allocate resources like CPU, memory, disk or network throughout the cluster. While the Resource Manager has a global stance within the cluster, the *Application Master* exists, temporarily and per-application, with the mission of coordinating the resource allocation between the Resource Manager and the several Node Managers. Finally, the *Node Manager*, existing one per each slave machine, is responsible for launching the containers that will serve the tasks' execution, while monitoring and reporting the resource usage to the Resource Manager.

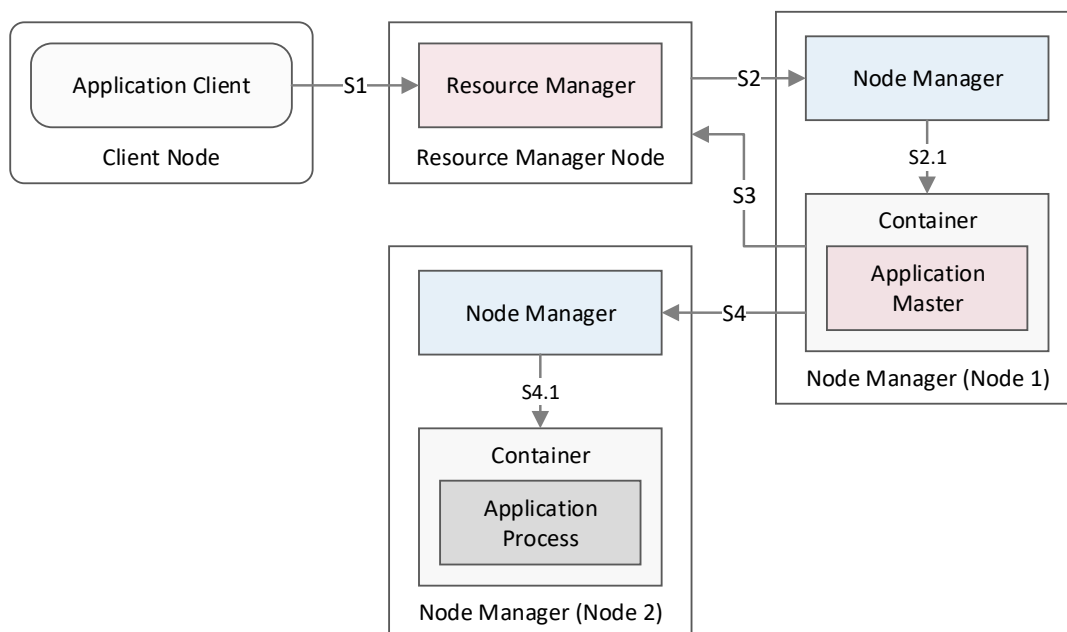


Figure 2.12. Application execution through YARN

In the figure above we can observe how the processing flow occurs since the moment a client requests resources for the execution of a data processing job (S1). The Resource Manager's first step is then to allocate an Application Master (S2 and S2.1) that will coordinate the job's execution. If the job requires more resources, so it can parallelize its execution, the Application Master requests for more resources to the Resource Manager (S3) and consequently more containers can be allocated in other nodes or in the same one. The diagram shows the allocation of a new container, with the purpose of executing a task, in a different node (S4 and S4.1).

YARN was designed to address many of the limitations present in Map-Reduce 1 and therefore includes many benefits like enhanced cluster *scalability*. With YARN, when compared to Map-Reduce 1, we can

have clusters up to five times larger, including a total of 10.000 nodes. The split between the Resource Manager and the Application Manager also brought to Hadoop higher levels of *availability*. Resources in YARN are fine-grained which allows for applications to request their customized allocation according to the specific tasks, thus reducing their wasteful usage. Unlike Map-Reduce 1, where we had a fix-sized allocation of slots pre-configured to serve Mappers or Reducers, the approach in YARN is much more flexible since that, instead of these static slots, we have a pool of resources that can be allocated dynamically according to the task's execution. Finally, the most revolutionary characteristic of YARN is its *multitenancy*. Only through YARN, it was possible to break from the fixed design that tied together the resource management and the processing engine. With this new architecture, Hadoop was made available to other frameworks and engines besides Map-Reduce.

2.7.4. Tez

Tez is a distributed execution framework directed towards data-processing applications. Tez itself, even though it appears as a new framework, does not break with the past. Many years of work and proven evolution regarding data shuffling operations and resource sharing in Hadoop and Map-Reduce are leveraged by Tez. The Map-Reduce paradigm is generalized, as a next evolutionary step in Hadoop 2 where the resource management is decoupled from the core through YARN, and Tez organizes its processing around the execution of complex Directed Acyclic Graphs (DAGs) of tasks representing the structure of a data processing workflow (Saha et al., 2015). Tez relies on the strengths of Map-Reduce and builds upon its weaknesses and constraints through the incorporation of techniques and strategies that can also be found in other frameworks, as in Microsoft's Dryad distributed execution engine (Isard, Budiu, Yu, Birrell, & Fetterly, 2007).

Tez is not an engine by itself, but instead a library that enables the creation of data-flow driven processing runtimes with, for example, Hive or Pig.

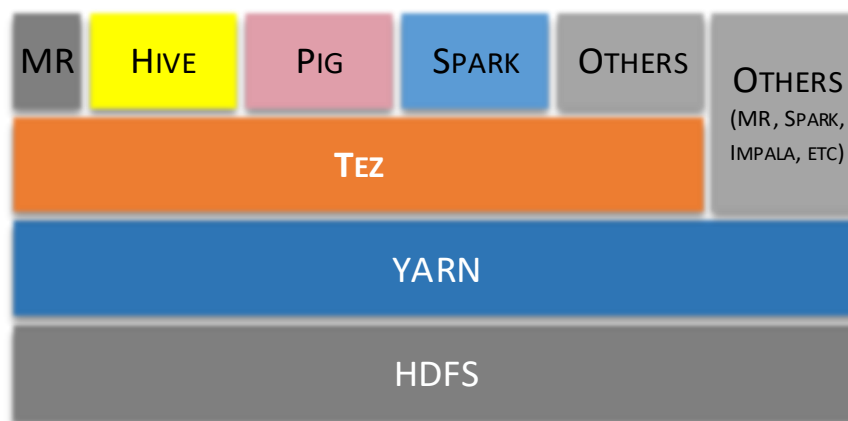


Figure 2.13. Hadoop 2 + Tez stack
(Adapted from Saha et al., 2015)

In Figure 2.13 we can see an extended version of Hadoop 2. There Tez is included above YARN, offering a framework to high-level engines like Hive, Pig or even Spark⁸. As depicted in the diagram above, Tez

⁸ Spark is an independent engine for Big Data processing that may be configured to use YARN, as a resource manager, and can also use Tez as its execution context.

offers an abstraction layer that negotiates directly with Hadoop's resource manager, YARN, but itself also provides an internal implementation of Map-Reduce that can be used for compatibility situations, even though engines should ideally use Tez as it offers great performance gains over Map-Reduce through, for example, a more efficient YARN container allocation or a more effective use of the HDFS (Singh & Kaur, 2016; White, 2015).

Apache Tez provides a unifying framework that facilitates and empowers the creation of purpose-built engines that can customize data processing specifically for their needs. Behind this, we can find three important aspects that characterize the architecture. First, the *expressiveness* of the model underneath Tez, captured by the representation of data movement through a DAG-oriented model, allows for a more natural definition of a broader set of computations. This approach is intended to break from the constraining model available in Map-Reduce where all the computations needed to be translated into Map and Reduce functions. The second aspect of the architecture of Tez is the *data-plane customizability* that allows for the definition of more specific semantics according to each different situation. When facing different algorithmic processes or different availability of resources, the adaptability of the processing model is fundamental for its efficiency. The semantics of Map-Reduce and its constraining task structure (Map and Reduce) make the customization difficult because it needs to intrude the engine's mechanics. Tez provides a lower level of abstraction that enables specific semantics and customizations, with the purpose of defining data transformations and data movements, more adapted to each particular situation. Finally, the third characteristic of Tez are the *late-binding runtime operations*. Hadoop clusters are, in nature, very dynamic environments that can be composed of heterogeneous nodes with completely different usage loads at different times and where their failure is accepted as normal rather than exceptional. With this in mind, Tez offers the possibility of applications to adapt their execution according to the data and to the environment through late-binding and on-line decision making. Things like partition cardinality or work division can be modified throughout the execution of a DAG so that they better fit into the cluster's characteristics at runtime (Saha et al., 2015).

As mentioned, Tez represents data movement through DAGs where data flows from data sources towards data sinks while being transformed in-between by intermediate vertices. DAGs, at a logical level, are composed of *vertices*, representing data transformations and *edges* that represent data movement.

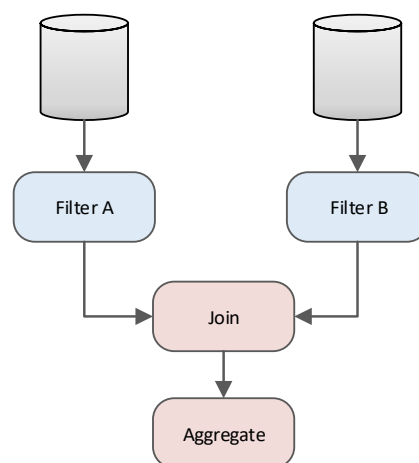


Figure 2.14. Logical DAG
(Adapted from Saha et al., 2015)

In Figure 2.14 we have an example of a simple DAG, at a logical level, showing two vertices (*Filter A* and *Filter B*) reading data from two data sources. This data is then processed by a third vertex (*Join*) through a *join* operation, and finally, a fourth vertex (*Aggregate*) aggregates this data. The edges, depicted as connections in the diagram, represent the movement of data between a producer and consumer vertices, and the vertices reflect the logical steps of data processing that apply rules to either filter or modify the data.

When we observe a DAG at a physical level, we can visualize its execution through individual tasks that are a composition of a set of *inputs*, a *processor* and finally a set of *outputs*. One vertex, through parallelism, can have multiple tasks responsible for processing different subsets of data from the same source.

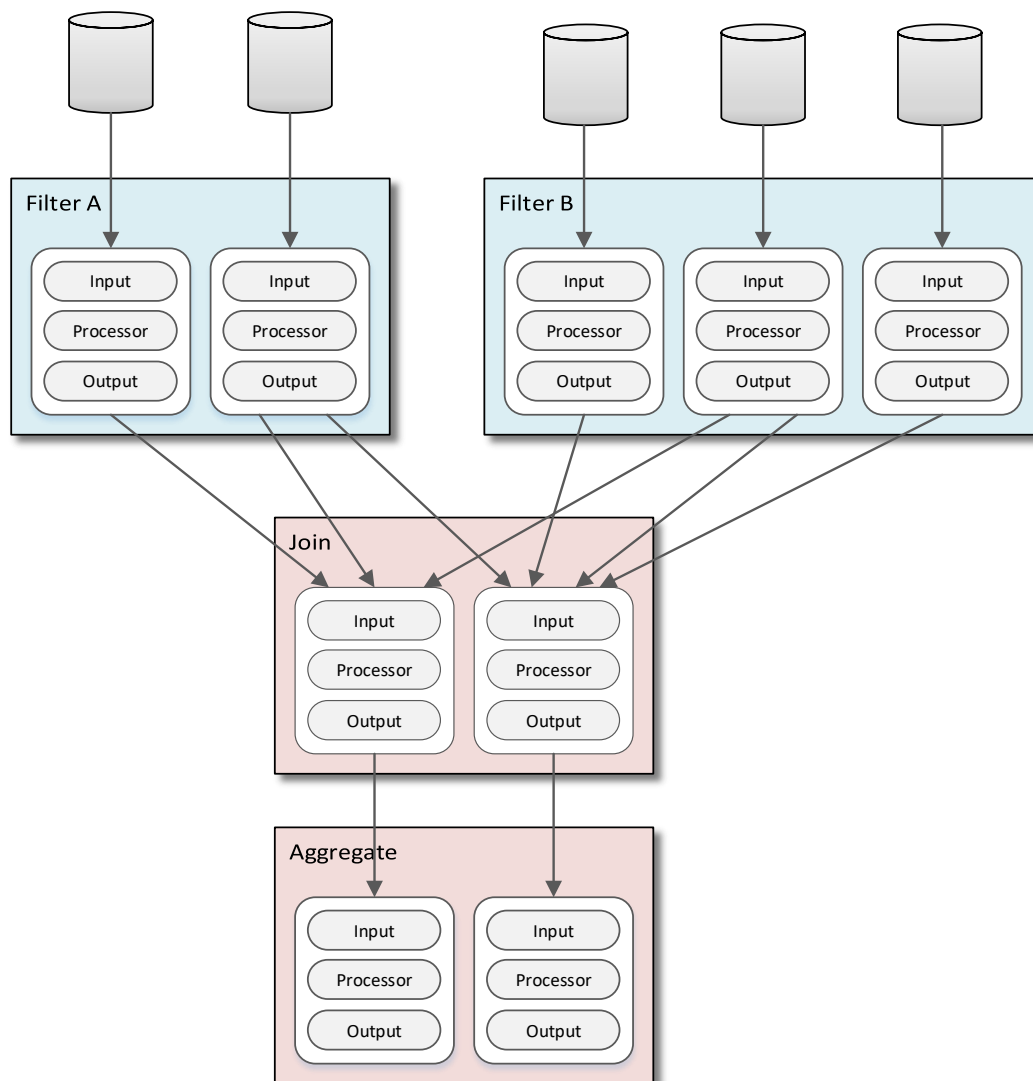


Figure 2.15. Physical DAG showing the actual execution
(Adapted from Saha et al., 2015)

In the figure above we can observe a DAG at a physical level where the vertices have multiple tasks and also the different properties of the edges expressing different data movements that can be *one-to-one*, *broadcast* or *scatter-gather*. In sum, the physical DAG is the set of tasks that are produced by expanding the vertices of a logical DAG into their corresponding tasks.

When constructing a DAG, each of the vertices can be associated with a vertex manager that is responsible to dynamically adapt the execution according to the characteristics of both the cluster and the data at runtime. It is the responsibility of the vertex manager to re-configure the vertices to maximize the efficiency of the DAG's execution.

Higher level applications, like Hive, build the data processing workflows on the fly with the Tez API library, by encoding their native language definitions into Tez DAGs. Afterward the DAG is submitted to YARN using Tez runtime library. The first step here is the creation of a new Tez Application Master designed to orchestrate the DAG's execution. This Application Master expands then the logical DAG to the physical level to include task parallelism and asks YARN for the required resources to perform their execution. Resources in the cluster, managed by YARN, are used in the form of containers. Each container represents the required resources (memory and CPU) to the execution of each individual task, that is a part of a vertex, and the Application Master itself also uses one YARN container. The tasks are usually executed according to the order defined in the DAG, and only the successful completion of all of them represents the completion of the DAG.

Tez, as a framework, enables a far broader expressiveness than Map-Reduce, and consequently this allows for higher level engines to more naturally fit and express their complex logic when defining data processing activities. Apache Hive was re-written to integrate with Tez and benefit from its major performance gains. One example of this are Hive's custom edges that are able to perform dynamically partitioned *hash joins* with the orchestration of a customized vertex manager. Runtime performance gains are also harvested from the generic execution efficiencies of Tez like, for example, the reutilization of containers.

2.7.5. Hive

Apache Hive provides a SQL-like translation layer on top of YARN where it is possible to define a data warehouse infrastructure over the data stored in the HDFS (Du, 2015). It enables data transformation and data analysis through the use of SQL while benefiting from the distributed processing paradigm implemented by Hadoop.

Like other translation layers in Hadoop's ecosystem, Hive intends to provide a more accessible way of writing data transformation algorithms when compared to Map-Reduce's programming model (Šubić et al., 2015), where, for example, to write the code to implement a simple *join* is not easy (White, 2015). In the world of data analysis, SQL is a widely popular and used language that has been around for more than thirty years. Hive provides then a high-level and more familiar programming model for data analysis and also eliminates some of the complexities inherent to Java programming (Rutherglen, Wampler, & Capriolo, 2012).

Hive is much more than a way of increasing familiarity and productivity in Hadoop. Hive's implementation takes on several performance improvements regarding data access, storage, and transformation. The HDFS flexibility, that enables it to store data in any structure, has its drawbacks performance-wise and Hive's implementation of structure over data in HDFS allows for its faster access. Hive also implements new efficient formats like the Optimized Row Columnar (ORC) file and relies on a Cost Base Optimizer (CBO), like several RDBMSs, to optimize query efficiency. Another

feature delivered by Hive is its vectorized query execution model that optimizes query execution at runtime by processing batches of 1024 rows instead of a single row at the time in operations like scans, aggregations, filters or *joins* (Huai et al., 2014).

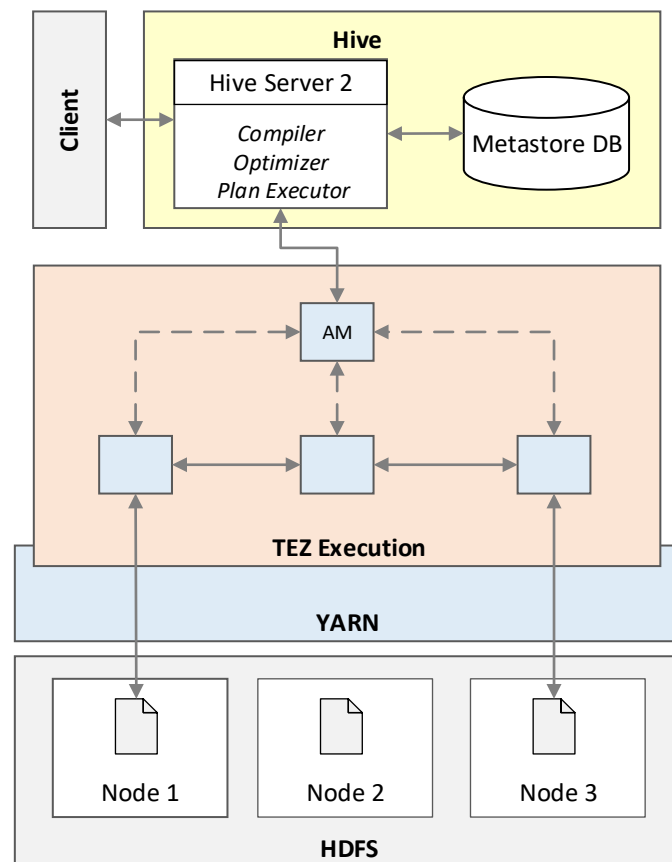


Figure 2.16. Hive query execution architecture with Tez

In the figure above we can observe the execution of a SQL query using Hive on Tez. The clients can connect to Hive Server 2 through JDBC⁹ or ODBC¹⁰ and execute their programs using Hive Query Language (HiveQL), Hive's SQL version. Hive's Metastore DB¹¹ contains all the information regarding the schema definition and also the statistics used by the CBO to determine the execution plans. With the information from the CBO, Hive optimizes the query execution and through Tez builds a DAG that is subsequently processed to extract and compute, according to the specified rules, the data stored in the HDFS.

Hive supports several structures, common in RDBMSs, like *tables*, *views* or *partitions* and a multitude of data types that can also include complex types like *arrays*, *structs*, and *maps*, to allow for a larger data variety. Despite this, Hive is not a database; Hadoop's design and purpose differ from what a relational database aims to provide. Hadoop's orientation for batch processing imposes that queries will have higher latency due to the overhead added by the creation of Map-Reduce or Tez jobs

⁹ JDBC stands for Java Database Connectivity and it is an application programming interface for Java that defines how a client connects to a database.

¹⁰ ODBC stands for Open Database Connectivity and it is a standard programming interface for accessing databases.

¹¹ Hive's Metastore DB is stored in an RDBMS using, for example, Derby or MySQL.

(Rutherglen et al., 2012). The concept of relationships between tables also cannot be expressed in Hive because referential constraints are not supported. Other differentiating aspects between Hive and RDBMSs, that were present in Hive's first versions, are being trimmed down. As of version 0.14, Hive supports transactions even though they have several limitations like, for example, being always auto-committed. More recently, in version 2.0, Hive introduced a new functionality called Low Latency Analytical Processing¹² (LLAP) with the purpose of lowering latency and providing interactive SQL to the users through caching mechanisms. From an initial approach that aimed to offer a more familiar and productive environment, so that data analysts could explore data in Hadoop, Hive is evolving towards a fully distributed data warehouse architecture.

As mentioned before, Hive offers a SQL-like language so that users can express their data transformation rules, the HiveQL. HiveQL does not conform fully with the ANSI SQL. It is instead a mixture of SQL-92, MySQL and Oracle's SQL (White, 2015). HiveQL also includes features from later SQL standards, like the analytical functions present in SQL:2003, as well as some specific features associated with distributed processing like the local *sort*. HiveQL is a language in constant evolution as new constructs are being implemented by the open-source community. Besides HiveQL, Hive, since its version 2.0, also includes a procedural language, the Hybrid Procedural SQL (HPL/SQL), that allows for users to add a new dimension of functionality to their data transformation activities and even to run code from existing RDBMSs on Hive, like PL-SQL from Oracle or Transact-SQL from Microsoft SQL Server. This heterogeneous and procedural nature of HPL/SQL enables it to implement ETL processes in an efficient way (Apache Hive, 2016).

2.7.6. Impala

Impala represents a new emerging class of SQL-on-Hadoop that, instead of relying on Hadoop's frameworks like Map-Reduce or Tez, uses its own engine to access and process data in the HDFS. Together with Hive, Impala is one of the most popular SQL-on-Hadoop systems that implement a database-like architecture. Both systems are part of the suites offered by the two major Hadoop distribution vendors. While the priority for Hortonworks is Hive, Impala is the SQL flagship for Cloudera (Floratou et al., 2014).

Impala was inspired by Google's paper where they present their project Dremel, an interactive ad-hoc query system for data analytics (Melnik et al., 2010). Impala differs from most Hadoop systems as its design is focused on optimizing latency and its architecture is similar to the ones we can find in traditional MPP data warehouses (Grover et al., 2014). Impala architecture uses its own long running daemons in every node, and each node can act as *Query Planer*, *Query Coordinator* and *Query Execution Engine*, the components of Impala's daemons. In Figure 2.17 we can observe these components during the execution of a query with Impala.

¹² Can also be referred as Live Long and Process.

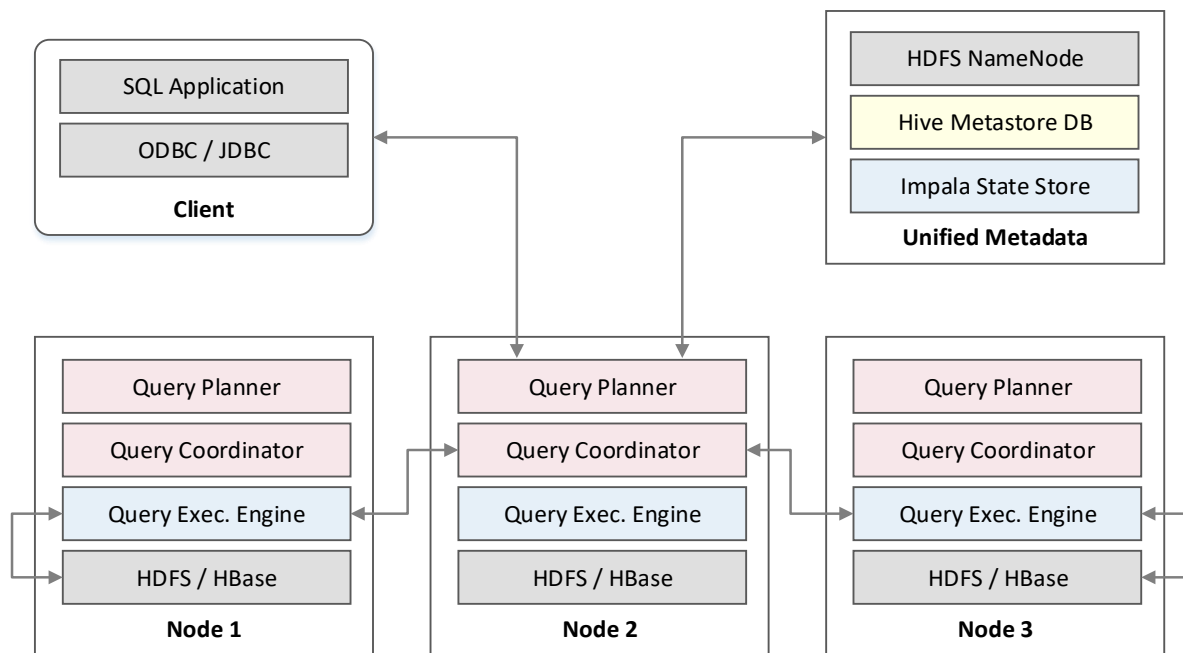


Figure 2.17. Impala query execution architecture

The figure above depicts the steps that are part of a query execution with Impala. Clients can connect to Impala, for example, through ODBC or JDBC and submit their queries that are parsed by the Query Planer. This component also produces the execution plan for the query, and this plan is then passed to the Query Coordinator that assigns parts of this plan to the several nodes in the cluster. Since each node hosts an Impala daemon, it is possible for them to execute the assigned parts of the query execution plan through the Query Execution Engine. Finally, when the processing is complete, the Query Planner returns the results to the Client. Note that in this diagram, under the Unified Metadata block, Hive’s Metastore DB is present since Impala was designed to use it so that we could avoid the creation of metadata silos within Hadoop’s ecosystem. Impala also uses the same SQL implementation as Hive, thus facilitating code and schema metadata sharing.

It has been stated that Impala offers a significant performance advantage over Hive, especially when the working set fits in memory (Floratou et al., 2014). Impala’s MPP architecture has several specific features that contribute to its performance enhancements. Impala’s completely re-written engine makes a more efficient use of memory because it is not restricted by the limitations of Map-Reduce, by, for example, caching data when the tables are scanned for the first time. The long running daemons, that are distinguishing characteristic of Impala, offer an advantage over other frameworks since they do not create any overhead due to startup because Impala is always running. Unlike many other applications in Hadoop, Impala is not written in Java. Instead, its engine was written from scratch using C++, thus offering several performance benefits, starting with the fact that it does not require a virtual machine, like the Java Virtual Machine (JVM), to access the hardware.

Still, on the performance chapter, there are several empirical works that prove that Impala is faster than Hive and, on the other hand, there are also demonstrations stating that Hive is faster than Impala. It is obvious that these studies are biased by whom is doing them – Cloudera demonstrates that Impala is faster (Cloudera, 2016) and Hortonworks argues that Hive on Tez is faster (Hortonworks, 2016a).

2.7.7. The SQL cycle

We have approached two of the most popular SQL-Like systems in the Hadoop ecosystem, Hive and Impala, but recently more interest towards the integration of SQL in Big Data triggered the creation of new SQL-like systems like Apache Drill and Presto or even the incorporation of a SQL layer in Spark, the Spark SQL.

If we look at the technological evolution within the Hadoop's ecosystem, and in a broader view even at the data analysis world, we cannot stop noticing an increased interest regarding SQL. The main problem behind the proliferation of Big Data was that the traditional DBMSs could not handle the amount and diversity of data that was being generated. This took us to solutions like NoSQL or Map-Reduce that broke with the relational and SQL paradigms. From this solution, another problem arose; Map-Reduce, in comparison to SQL, is a lot more complex to use and consequently the implementation of data transformation and analysis processes became more difficult and more time consuming to the developers accustomed to SQL. From this point forward, SQL started to be integrated with Map-Reduce, and even though this simplified the implementation process, we were still trapped by the constraints of Map-Reduce and its batch-oriented processing with poor performance for interactive querying. So finally, what we are observing is that the interest towards high-performance massively parallel processing SQL engines is growing. In summary, the Big Data technologies that initially broke from SQL are finding their way back into it (Chandran, 2013; Floratou et al., 2014).

2.8. HYBRID APPROACH

At their core, RDBMSs and Hadoop were created to deal with the same root problem, data management. Their features, weaknesses, and strengths are quite different as it were the circumstances from which both these technologies emerged. If we can argue that RDBMSs are no longer capable to cope with the volumes and variety of data that is produced nowadays, we can also argue that Hadoop is still too young and too volatile to be seen as a reliable technology to play such a critical role in the decision-making process that is nowadays fundamental in modern companies. Discussions aside, it is recognized that both technologies have their merits and can be seen as complementary in the evolution of data warehouses (Russom, 2014).

The technological landscape around data management is constantly evolving. Massively Parallel Processing databases, Columnar data stores, NoSQL approaches and Hadoop's vast ecosystem are just examples of how technology can help us extract value from data. Deciding on which path to take is perhaps the hardest step. The challenge of creating the best Data Warehousing architecture is now more complex than ever due to the plethora of solutions and technologies available (Clegg, 2015).

Starting from the educated belief that Enterprise Data Warehouses are and will continue to be extremely valuable for the companies' decision-making processes (Kimball & Ross, 2013; Krishnan, 2013), how can we then expand their capabilities and consequently provide them with the ability to adapt to their ever-changing environment and through this assure their viability (Carvalho, 1998)? The list of opportunities and alternatives is vast, and of course, they need to be weighted according to the specific problems at hand. One path to be explored, the one being assessed in this research, is the use of Hadoop as a transformation tool and this presents itself as great opportunity for organizations to

maximize their current EDW infrastructure through the process of offloading the massive amounts of data being stored and processed currently by their data warehouse (O'Reilly Media, 2015).

In some aspects, Hadoop outperforms RDBMSs and vice-versa, so it is valid to equate that, by combining them together, each of them can minimize the limitations of the other and potentially result in a fruitful synergy (Russom, 2014). To support this hypothesis, the first step is to further the understanding of both technologies in a context where their combination can be used to produce a superior Data Warehousing architecture.

A DW can be designed to support reporting or more advanced analytics and, of course, to support the combination of both, but it is important to discern its architecture so that we can better understand the aspects that can be improved by the use of Hadoop or an RDBMS. A data storage, supporting a reporting layer, demands above all low latency. Users want to access and visualize reports in seconds, and relational databases excel for this purpose. Another advantage in RDBMSs, when supporting a reporting layer, is the implementation of the data structure. When serving reports, data needs to be transformed and clearly structured and Hadoop was not designed to enforce structure or define relationships between the entities. This is what relational databases have been providing for more than forty years. If it is true that reporting requires structured data, that does not mean that the source is mandatorily structured. Data warehouses collect data from many and diverse sources, and this data requires treatment before it can be incorporated in the data warehouse. For that purpose, the transformation of large and diverse datasets, Hadoop is better suited for the task. Its design is oriented for distributed batch processing that can easily scale to better accommodate distinct amounts of data (Hortonworks, 2016b).

Data warehouses represent a unified version of the truth that characterizes an organization at a given time and also keep a record of its history. Hadoop, as cheaper and more scalable solution, can be used to store, and automatically compress, the “cold data”, older data that is rarely accessed by the reporting layer. This offload alleviates the data warehouse, not only in storage requirements but also in processing needs (Hogan & Jovanovic, 2015). The huge amounts of data stored in a Hadoop cluster can also be easily accessed by business analysts, at any given time, with the purpose of performing ad-hoc analysis without having to rely on pre-defined ETL processes (Duda, 2012).

In summary, an RDBMS is an excellent option to store facts, dimensions and aggregated metrics that are used to support the reporting layer, while Hadoop is a great fit for ETL, data storage and to serve as a platform for knowledge discovery and advanced analytics (J. A. Lopez, 2012). This offload of data and time-consuming processes to Hadoop is a cost-effective and non-disruptive approach of incorporating the emerging technologies from the Big Data landscape in the architectures of traditional Enterprise Data Warehouses (O'Reilly Media, 2015).

2.9. IPTV

IPTV is a protocol that allows for the delivery of media content through Internet Protocol (IP), rather than through broadcasting like satellite, terrestrial or cable. With IPTV is possible to deliver a wide variety of contents beyond the traditional live television. Users can stream videos, visualize live television replays, use applications or even play games. Media content is streamed from the IPTV

infrastructure to the customer premises over different network technologies like xDSL (Digital Subscriber Lines) or GPON (Gigabit Passive Optical Network) and it is received in user equipment's like the Set-Top Boxes (STBs).

Ericsson's Mediaroom is a middleware composed by a collection of software that allows for service providers to deliver the IPTV functionalities like Live Television (TV), Digital Video Recording (DVR), Video-on-Demand (VoD) or Interactive TV Applications. Mediaroom was originally created by Microsoft in 2007 but in 2013 it was acquired by Ericsson.

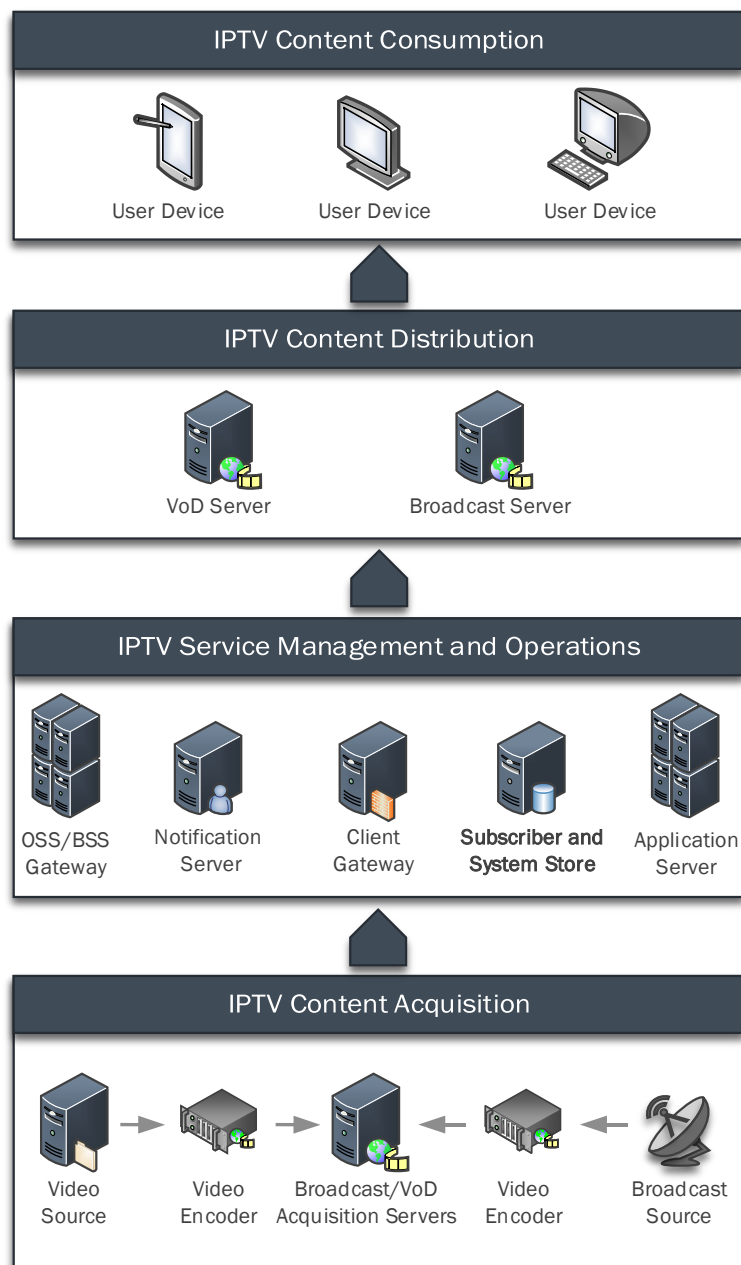


Figure 2.18. IPTV high-level architecture

At a high-level view, we can discern the IPTV platform in the four major areas presented in Figure 2.18. Firstly, the different types of contents (Live TV and VoD) are acquired by the Acquisition Servers and, according to the configurations defined in the Operations Support Systems (OSS) and Business Support Systems (BSS), are then distributed to the end users (Architecture of Microsoft Mediaroom, 2008).

In this study, we are interested in collecting the metadata and information stored by the OSS and BSS (Ericsson Mediaroom Server Operations Help, 2015). Service, device and channel configurations, content metadata, subscriber information and ultimately the data regarding the user behavior, needs to be extracted from the operational systems and integrated into a data warehouse that, through a set of transformation processes, is capable of providing tangible and valuable measurements concerning television audiences.

Analyzing IPTV and Mediaroom themselves is not the main goal of this study, they represent just an instance of the main problem with which we drive our research. Nevertheless, tearing apart and comprehending the IPTV infrastructure, at a logical level, is fundamental to understand the entities and relationships between them, that all put back together, in a database model, will allow us to analyze all the available data and from there generate useful insights.

2.10. TELEVISION AUDIENCE MEASUREMENTS

Even though the focus of our research is not on the topic of audience measurements, we recognize its importance and impact in the television industry as they support the operation of commercial media (Nelson & Webster, 2016). Like many other businesses, advertising is no foreigner to the new technological trends. Big Data technologies play an important role in analyzing the large data sets generated by many kinds of audiences not only from the traditional medias, like radio and television, but also from the new universes of digital content and social media (Marks, 2013).

Audience research can be conducted through several approaches, that are motivated by different goals and methodologies, and can use both qualitative or quantitative methods (Webster, Phalen, & Lichty, 2014). These quantitative methods can rely on content analyses, with the purpose of quantifying data, that can be used afterward by more in-depth researches. As mentioned, our study is not an audience research, but instead, it can be seen as an enabler of future audience research initiatives because it is able to generate relevant quantitative measurements from the analysis of audience behaviors. From the specialized literature, we highlight three measurements that are commonly used in audience research projects (Webster et al., 2014):

- *Rating*: the average of a given population watching a television channel or a program across a time interval (can also be expressed as a percentage calculated over the total number of potential households);
- *Reach*: the cumulative percentage or total of a population that watched a given channel at least once during a time interval;
- *Share*: the percentage of individuals that viewed a given channel or program during a specific time interval, calculated over the total number of individuals watching television during the same period.

These are the main measurements that drive the design of the data models implemented during our study, but other metrics could be generated to support a comprehensive audience research project.

2.11. SUMMARY

The core of the theoretical framework supporting our research relates to data science and how data can be transformed into ultimately actionable knowledge. We have explored the origins of relational databases and highlighted their still critical role in supporting data warehouses and their architectures. Invariably driven by the constant growth of data, data warehouse architectures are forced to evolve as relational databases struggle to transform data into usable information and here we have explored the emerging technologies from the universe of Big Data, namely the Hadoop's ecosystem and more specifically the SQL-like approaches that offer the proven advantages of SQL while implementing scalable distributed processing and its ability to handle enormous amounts of data.

Finally, our theoretical framework briefly touched the architecture and functionalities of Internet Television and how the data extracted from this infrastructure can be transformed into valuable audience measurements. Even though audience measurements are not the core of our research, their importance and the challenges they create due to the amount of data generated, represent a specific situation that triggers the need to evolve data warehouse architectures to assure their relevance and viability.

3. METHODOLOGY

3.1. INTRODUCTION

Research is a systematic process, supported by data, through which we attempt to answer a question, solve a problem or expand our understanding of a phenomenon (Leedy & Ormrod, 2010). Research methodologies provide frameworks and guidelines in the definition of the research scope, how it should be performed and what type of conclusions can be inferred from the data that is collected (Williams, 2007).

In order to set the appropriate methodological path linking our initial problem, and related aspects, to the goals of our research, we approach the definition of its underpinning methodology in this chapter. The design of our research was driven by the phased approach proposed by Saunders et al. (2016) known as “The Research Onion” and complemented by the Design Science methodology (A. Hevner & Chatterjee, 2010) in the aspects concerning the methods and strategy.

3.2. RESEARCH DESIGN

The research design defines, at a high-level, the plan and the steps put in place so that initial research questions can be answered (Saunders et al., 2016). In our study, the elaboration of this plan was driven and supported by Saunders’s “Research Onion” (see Figure 3.1).

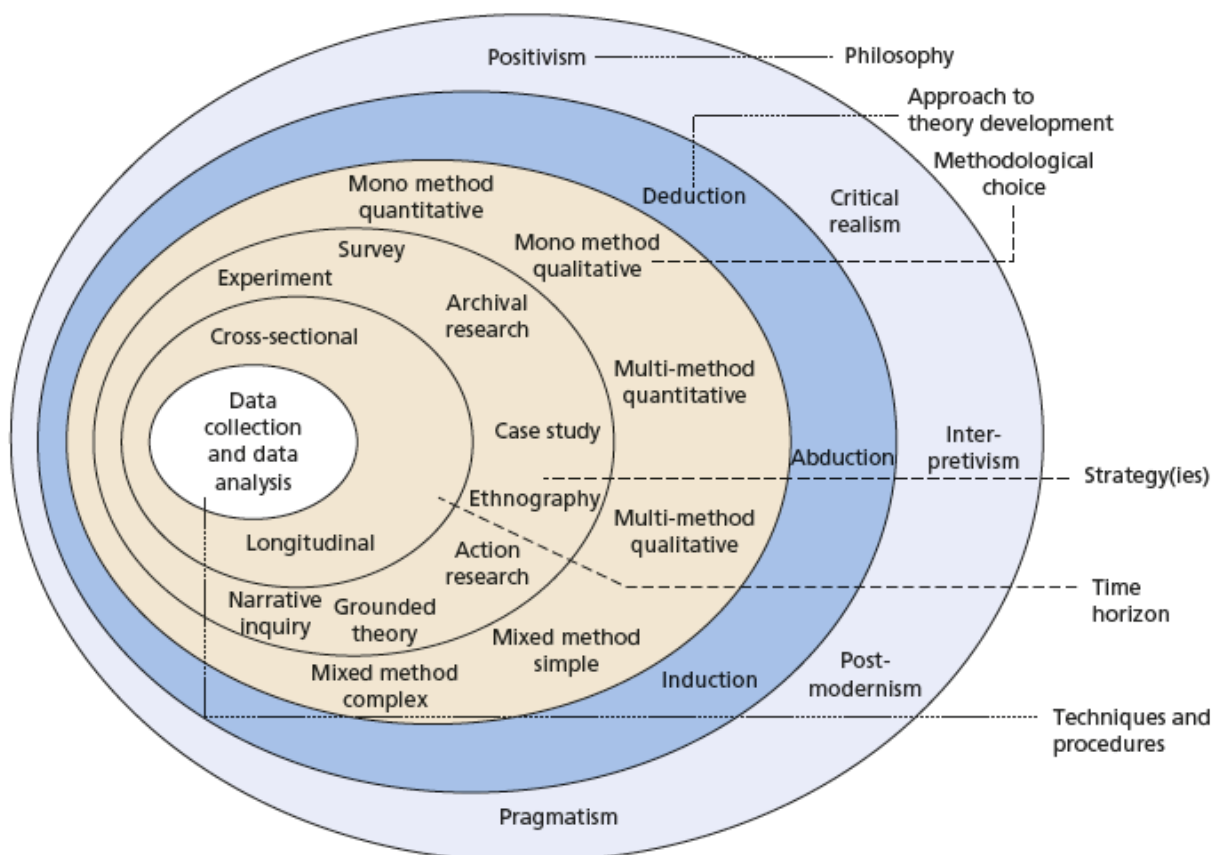


Figure 3.1. The Research Onion (Saunders et al., 2016)

The Research Onion consists of a set of layers, or research dimensions, that guide the researcher to methodically define each of the steps that must be taken to effectively achieve the research goals. Within each layer, we are asked to perform a set of questions that help frame and direct the research and consequently define the concrete steps required to implement it. In the next sections, we go over the different research dimensions and see how they directly relate to our research.

3.2.1. Research philosophy

The first layer, proposed by Saunders's Research Onion, is concerned with the philosophical aspects of the research. This research dimension relates to the researcher's view of the world, its assumptions and the nature of the realities studied (Saunders & Tosey, 2013). A thorough analysis of the underlying philosophical aspects of the research is critical to define how the research questions are understood, which methods to use and how the findings are interpreted (Crotty, 1998). To achieve this definition, researchers perform a set of claims, under the viewing of scientific philosophy, regarding: the view of the nature of reality (*Ontology*), how we know it and what is acceptable as knowledge (*Epistemology*), what values are associated to it (*Axiology*) and the methods applied to its study (*Methodology*) (Creswell, 2003).

According to literature, the most referenced schools of thought, or research paradigms, are the *Positivism/Postpositivism* and the *Interpretivism/Constructivism* (Easterby-Smith, Thorpe, & Jackson, 2012). *Positivism* is an empirical approach that relies on the idea that reality exists independently of the study, meaning that it is possible to observe and describe reality from an objective view point (Bryman, 2013). *Interpretivism* advocates that the meaning of the phenomena is created by the observer and, therefore, it is subject to interpretations that are bounded to the observer's values (Creswell, 2013). There are other research paradigms, but we devote special attention to the ones that could be aligned with our research, like *Pragmatism* that conceives knowledge as an experiential process rather than a reflex of an independent reality. Pragmatism is a philosophy where the worldview is built from actions, situations, and consequences (Creswell, 2013; Morgan, 2014). To relate the research paradigms to the different philosophic views, and through that define which is better suited to our research, we present, in Table 3.1, their main characteristics.

Philosophical View	Research Paradigm		
	Positivism	Interpretivism	Pragmatism
Ontology	External; Single reality; Independent of social actors and objective.	Socially constructed; Multiple realities; Subjective.	External; Multiple realities; Adaptable to better answer the research question.
Epistemology	Objective – only observable phenomena provide credible data and facts; Focus on causality and generalizations, reducing	Subjective – social phenomena and subjective meanings provide acceptable knowledge;	Objective and/or subjective – observable phenomena and/or subjective meanings can provide acceptable knowledge;

	phenomena to their simplest elements.	Focus on the details of situations and the subjective meanings motivating the actions.	Focus on practical research and integrates different perspectives to interpret data.
Axiology	Objective; Independent from the researcher's values.	Subjective; Highly dependent on the researcher's values (researcher and research cannot be dissociated).	Researcher can adopt both objective and subjective stances where values have a late role in interpreting the results.
Methodology	Primarily quantitative (can also be qualitative); Large datasets; Measurement gathering.	Qualitative methods; Small datasets; In-depth detailed investigations.	Both quantitative and qualitative methods; Mixed or multiple method designs.

Table 3.1. Research paradigms according to philosophical views
(Adapted from Creswell, 2013; Gregg, Kulkarni, & Vinzé, 2001; Saunders et al., 2016)

Defining the research paradigm is essential to clarify the nature and goal of our research as it helps to establish a clear and methodological route between the initial questions, their rationale and ultimately their answers and contributions (Saunders et al., 2016). By reflecting upon the characteristics of the pertinent research paradigms to our study, presented in Table 3.1, we are able to establish a parallel with our own research and define Positivism as the chosen research paradigm for our study. We start from the hypothesis that Hadoop can be used to enhance the shortcomings of traditional data warehouses, when dealing with large amounts of data, and to validate this theory, we conduct a series of experimentations on a specific problem, the calculation of television audience measurements. The experimentations will produce a series of quantitative data that will be used to validate the initial hypothesis and afterward, from the established facts related to the specific problem, our goal is to move to the generalization of our conclusions towards the definition of an enhanced data warehouse architecture that incorporates Big Data technologies.

3.2.2. Research approach

The second layer of Saunders's Research Onion relates to the types of approaches that can be adopted during a research. The reasoning behind the research is driven by the relationship between theory and the research itself; whether we move from the theory to the observations or if are the observations that allow us to form the theory, thus the *deductive* and *inductive* researches (Bryman, 2013). With *deductive reasoning* we arrive logically at the conclusion from an initial set of premises, being the conclusion only true if also the premises are validated positively (Ketokivi & Mantere, 2010). In *inductive reasoning*, the results of the observations can be used to produce premises that are then used to support the validity of the conclusion (Saunders et al., 2016). While in a deductive approach the researcher defines a hypothesis from the theory and then proceeds to validate this hypothesis, in an inductive approach the outcome is the theory itself that is inferred from the conclusions (Bryman, 2013). There is also a third research approach, the *abductive*, that is devoted to the explanation of "surprising facts". These facts are defined as the conclusions, and the purpose of the research is to determine a set of premises that are able to explain them (Ketokivi & Mantere, 2010).

Our research evolves in a single continuum, but it is composed of two distinct moments that are driven by two different reasoning approaches. From our initial hypothesis, based on the premise that Hadoop can be used to augment a traditional data warehouse that lost its viability due to the volume of data, we follow a deductive reasoning approach that drives us from the theoretical exploration and subsequent empirical experimentation to the positive validation of the hypothesis and resolution of the specific problem. Afterward, from the results gathered in our experimentation phase and the insights gained during the theoretical study, our research is driven by an inductive reasoning with the purpose of generalizing our conclusions to a broader area, the evolution of Data Warehousing architectures through the inclusion of Big Data technologies.

3.2.3. Research strategy

Moving into the inner layers of Saunders's Onion, we are confronted with the choices regarding the design of the research. Instead of analyzing the next two layers separately (Methodological Choice and Strategy) we believe that, for our study, it is useful to evaluate them together. A strategy is typically based on science, and it is comprised of a coherent relationship between a goal, a sequence of steps to be followed and a set of techniques associated with these procedures (Saunders & Tosey, 2013). Strategy and methods are closely connected to the support of the underpinning research approach.

As mentioned previously, our research is composed of two stages driven by different approaches, and thus the methods associated are also distinct, even though they complete each other. The first stage attempts to validate our initial hypothesis mainly through the collection of quantitative results gathered from the experimentations, while the second phase aims to generalize our results through more qualitative techniques like observations and insights inferred from our literature analysis. The mixed nature of the methods and their sequential combination gives a clear indication that the most appropriate design for our research are the *explanatory sequential mixed methods* (Creswell, 2013).

Defining an adequate research strategy does not serve the purpose of constraining the research body but instead serves to direct it methodologically towards its goals (Saunders & Tosey, 2013). The strategy definition does not impose on the research but instead facilitates it, as it is devised primarily from the consideration of the problem at hand, the related body of knowledge and the available data (Ellis & Levy, 2010).

Most of the research in Information Systems follows two distinct paradigms – *Behavioral/Analytical Science* and *Design Science* (A. R. Hevner, March, Park, & Ram, 2004). The first is more concerned with the observation of IS as a phenomenon, thus focusing on their characteristics and on predicting computer-human interactions, while the design-oriented paradigm aims to develop and provide guidelines for the construction and operation of IS and also for the creation of innovative concepts within Information Systems (Österle et al., 2011). Considering the goals, nature and the body of knowledge of our research, the design-oriented approach was selected as the guiding methodology since it focuses on understanding, explaining, improving and innovating Information Systems (A. Hevner & Chatterjee, 2010). In alignment with the Design Science methodology, we identify and describe the associated activities and their application to our study in Table 3.2.

Activities	Application in the research
Problem identification and motivation	<p>Our initial problem comes from the necessity of evolving a data warehouse responsible for the transformation of Mediaroom subscriber events into relevant television audience metrics. This DW, due to the amount of data, is no longer capable of executing the proper data transformations within an acceptable timeline and thus it fails to deliver the required information. Beyond the resolution of this immediate problem, our research is motivated by the study of Big Data technologies and their application with the purpose of augmenting Data Warehousing architectures.</p>
<i>Knowledge¹³ Acquisition</i>	<p>The theoretical foundation of our research focuses on two major areas of interest – Data Warehousing and Big Data. To understand data warehouses, we performed a study not only of their architectures but also of their underlying technologies and models. This study was then expanded by an assessment throughout the solutions within the Big Data ecosystem so that theoretical foundations, architectures and technologies, from both worlds, could be combined most effectively and efficiently towards the resolution of our problem.</p>
Define the objectives for a solution	<p>Our objectives are defined according to our initial problem and the possibilities and limitations associated with the relevant body of knowledge. In a broad view, our research is comprised of two main objectives:</p> <ol style="list-style-type: none"> 1. Implement a technological solution capable of generating television audience measurements from the raw data produced by Mediaroom and presenting them to the users; 2. Propose an enhanced DW architecture where Hadoop is the main actor performing the data transformations. <p>To achieve these general objectives our research included other more detailed objectives that were described in section 1.2.</p>
Design and development	<p>The main construct of our research is the DW that produces and presents the TV audience measurements. To achieve this, the reality of the problem was studied, modeled and implemented in both an RDBMS and a Hadoop cluster. The dual implementation is related to the assessment of how Big Data technologies can, in fact, enhance data warehouse architectures. Our research also delivers another artifact that is not an IS but rather our considerations towards the definition of an enhanced DW architecture that combines Hadoop with an RDBMS.</p>
Demonstration	<p>The generated TV audience measurements are presented by means of graphical reports and dashboards.</p> <p>The validation of the initial hypothesis, that argued that Hadoop could be used to enhance traditional data warehouse architectures, is demonstrated through the presentation of comparative benchmarking results.</p>

¹³ This activity, according to the proposed six steps of Design Science (A. Hevner & Chatterjee, 2010), is part of the “Problem identification and motivation” step, but we believe that, due to its importance to our research, its best highlighted as a separate step.

Evaluation	The evaluation of both the set of processes that generate the audience measurements and the usability of Hadoop as a transformation tool was performed by several benchmarking tests regarding performance and storage. The same performance tests were also executed on a subsequent iteration designed to evaluate the scalability potential of the systems.
Communication	The main target audience of this research are IS solution architects and managers looking to enhance their data warehouse architectures. To reach this audience, we will produce a paper with our summarized results and conclusions.

Table 3.2. Activity summary according to the Design Science steps

The table above describes summarily the activities that compose our research. These activities, and globally our research, are much in line with the essence of Design Science, as it is problem driven, literature based and it is focused on the production and evaluation of artifacts (Ellis & Levy, 2010).

3.2.4. Time horizon

The time horizon dimension is not particularly relevant to our research because our focus is towards the problematics surrounding the transformation of large volumes of data, rather than the extraction of knowledge from a specific dataset. The data used in our experimentations reflects several days of television users' behaviors but the days themselves have no relevance for the study. The results and conclusions of our research are not time-dependent.

3.2.5. Data collection

Our research uses large amounts of quantitative data. This data reflects television users' behaviors, and it is directly extracted, in bulk, from the Mediaroom platform. Data is vital to our research, but our concern is towards the impacts of its volume rather than the behaviors extracted from its content. For this reason, data collection is a simple and straightforward task, even though we have a set of steps to prepare the data, before its utilization during our experimentation phase, with the purpose of assuring its quality and facilitate the benchmarking tests.

3.3. SUMMARY

This chapter addressed the importance of methodology in establishing the views, boundaries, and techniques that are part of any research project and later instantiated them to our research, with the purpose of clearly defining the steps that took us from the initial problem to the materialization of the goals. In Table 3.3 we present the summary of the methodological dimensions and their instantiations in the context of our research.

Dimensions	Instantiation in the research
Philosophy	Positivism
Approach	Mainly deductive but complemented by inductive
Strategy	Methods and strategy are defined under Design Science
Time horizon	Not applicable
Data collection	Primarily quantitative data extracted from the Mediaroom platform
Target outputs (Artifacts)	<ol style="list-style-type: none">1. DW capable of producing TV audience measurements2. Hybrid DW architecture proposal3. Installation guides for an Oracle DB and a Hadoop cluster (Annexes)

Table 3.3. Methodology summary

4. DESIGN AND DEVELOPMENT

4.1. INTRODUCTION

Throughout this chapter we will describe the practical aspects that supported and materialized our study. From the identification of the problem, we searched for solutions and explored them, as described in the Theoretical framework chapter, and here we materialize the knowledge previously acquired so that not only the initial problem is tackled, but also new insights can be gathered regarding the inclusion of Big Data technologies in data warehouse architectures.

We start by defining the boundaries of the problem and devise a feasible solution that consists of an enhanced data warehouse architecture. The validity assessment of the proposed system requires empirical substation and, for that purpose, one of our first steps is to implement the data warehouse that relies solely on a traditional RDBMS. This is only possible after a comprehensive analysis of the IPTV infrastructure and especially how Mediaroom manages it. It is this analysis that allows us to design the DW data model and subsequently implement all the required processes that will transform data into usable information, the audience measurements. Before the implementation itself, we have installed and configured the environments that would support it. It is then, in both these systems, the RDBMS and the Hadoop cluster, where the transformation processes are effectively implemented.

In Table 4.1 we present the main steps, framed under the “Design and development” activity of Design Science, which are executed in this chapter.

Step	Phase	Description	Section(s)
1	Design	Problem description and high-level analysis	4.2, 4.3
2	Design	Entity identification and data preparation	4.4
3	Design	Data warehouse design and modeling	4.5
4	Development	Environment preparation	4.6
5	Development	Data warehouse implementation in the RDBMS	4.7
6	Development	Data warehouse implementation in Hadoop	4.8

Table 4.1. Design and development activities

4.2. PROBLEM DESCRIPTION

The specific problem being tackled in this study portrays a practical case of a traditional data warehouse system that simply can no longer produce answers due to the increase of data it had to rely upon. A data warehouse that processes television viewing events, reflecting the users’ behaviors, and transforms them into useful insights for the business area, has a critical value for any television service provider, but for the system to maintain its validity, it has to adapt to the increase of data it has to process.

Customers of a television provider using the Mediaroom platform perform actions like changing a channel, recording a program or visualizing a video, at their homes, using the set-top box (STB) remote control, and these actions are stored in a central repository. These events are afterward collected and transformed into meaningful metrics that are made available through reports and dashboards.

4.3. DATA WAREHOUSE ARCHITECTURE

From the IPTV infrastructure, and more precisely from the data generated by the Mediaroom platform, the data is extracted as text files and loaded into the data warehouse where it is transformed to meet the reporting needs. This architecture, portrayed in Figure 4.1, uses an Extract-Load-Transform (ELT) approach where all the transformations are performed inside the database. This approach makes the data warehouse architecture less complex since no extra transformation tools are needed, but it requires more processing power on its database. Advantages and disadvantages of this architecture were described in section 2.5.1.

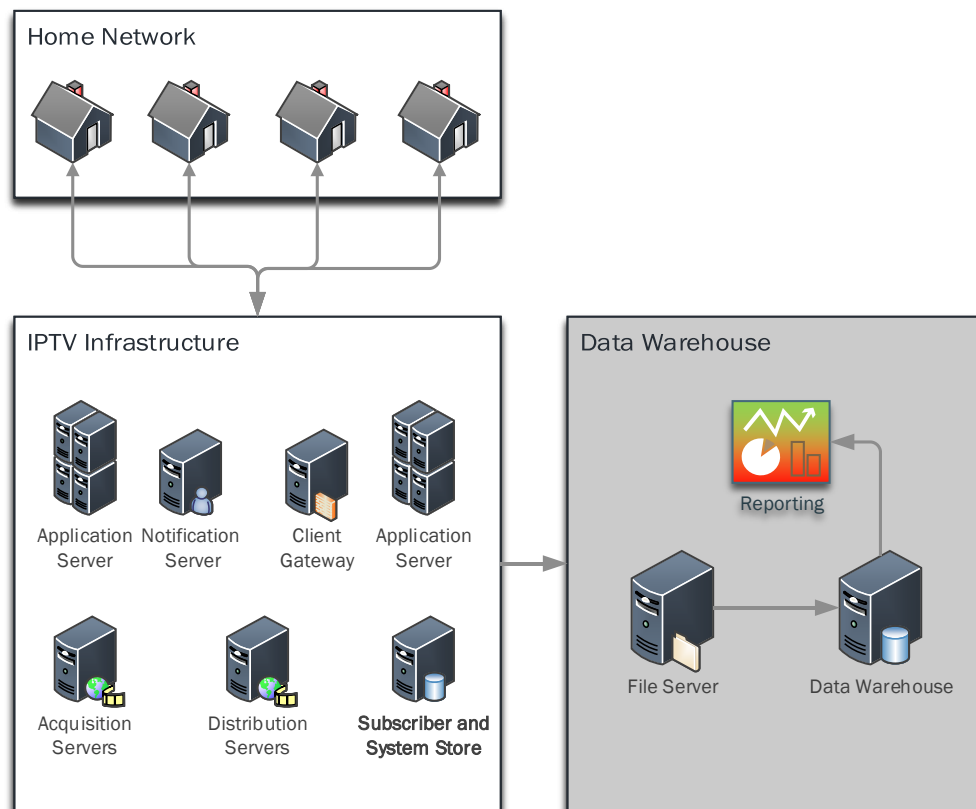


Figure 4.1. Current data warehouse architecture

Scalability on a traditional Relational Database Management System has severe constraints since it is vertical. If more processing power is required, mandatorily we need to enhance the resources and capabilities of the processing unit, in this case, the database. Coping with large volumes of data that require great processing capabilities is a problem that is only exacerbated by the ELT approach. With ELT, the transformations are being performed in a single place whereas in an Extract-Transform-Load approach it is possible to, in many cases, break down these transformations.

We are faced with the problem where, due to the increasing volume of data being fed to the data warehouse, we can no longer produce useful information on time. Upon such scenario, several directions could be followed. The simplest, but also the most expensive, would be to resort to the enhancement of the database capabilities and upgrade the database server, but this solution could become obsolete very quickly if the volume of data continues to increase at a quick pace.

Breaking down the transformation processes and moving them out of the database, thus changing the paradigm from ELT to ETL, is also a viable solution for the problem at hand. Nevertheless, this approach would increase the complexity of the data warehouse architecture and consequently its management and operation. Another important aspect that we need to take into consideration is the effort required to re-write the transformation processes that are currently in the database, according to the adopted transformation tool.

Finally, and taking into consideration very specifically the processes involved in this study, we need to account for the analytical component of the system. Even though we need to load and transform enormous volumes of data, we cannot overlook the processes that will consume this data. The system relies on heavy aggregation processes, built upon hundreds of millions of records, which will ultimately produce the desired metrics.

Even though it is feasible to move the transformation processes out of the database into a specialized transformation tool, doing the same for the aggregations would not be as feasible. Aggregations rely on gigabytes of transformed and enriched data that is stored inside the DW and, therefore, just the need to extract this data out of the database would represent an expensive and time-consuming step. Another important aspect is the concept of aggregation itself that makes it harder to manually decompose into smaller processing tasks.

Faced with these constraints when dealing with enormous amounts of information, this study focuses on incorporating a Hadoop cluster in the data warehouse architecture. The DW will consume all the data transformed and aggregated by the cluster and make it available to the reporting layer. This hybrid approach will allow for a flexible architecture capable of adapting to the effort required by the transformation processes that can vary throughout time or if the project is applied on a different scale.

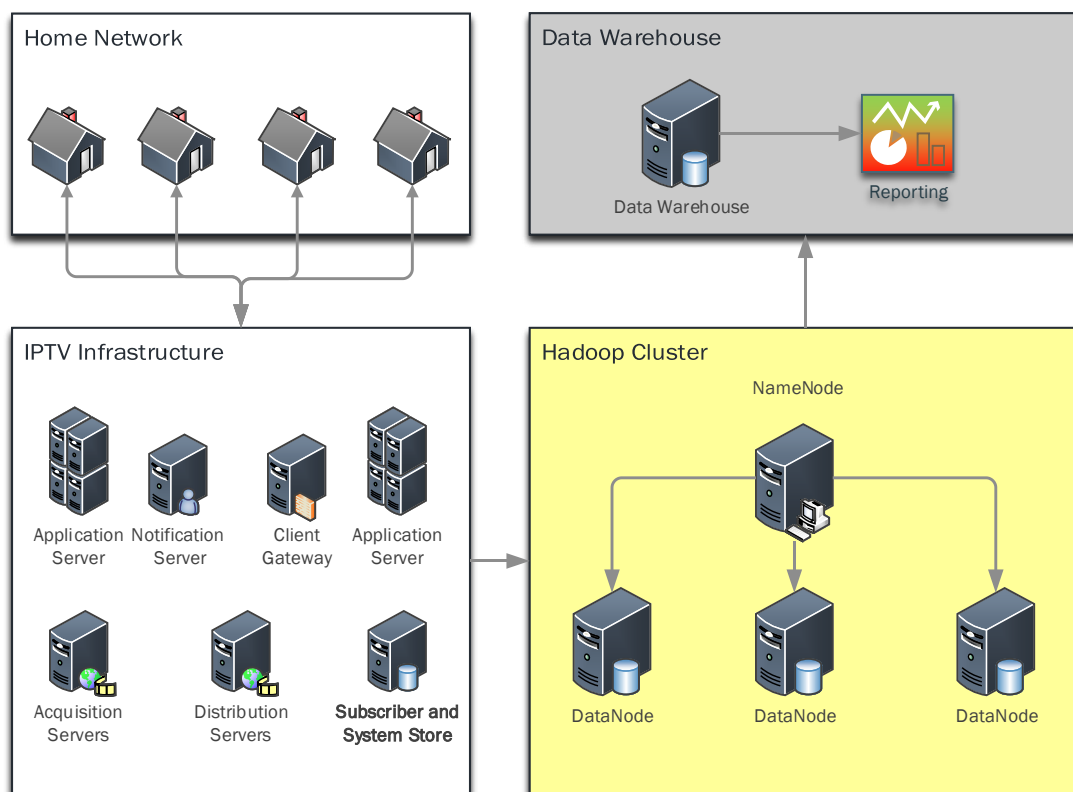


Figure 4.2. Proposed data warehouse architecture

Combining a data lake¹⁴, empowered by the processing capabilities of a Hadoop cluster, with the RDBMS supporting the already established reporting layer will add a transformation layer that can easily adapt to the volumes of data. The Hadoop cluster, shown in Figure 4.2, will also serve as a repository, replacing the file server on the previous architecture (Figure 4.1), where historical data can be stored and used by future ad hoc analyses that go beyond the information provided by the established reporting layer.

4.4. SOURCE DATA

The source data of this study is produced inside the IPTV infrastructure and, more specifically, by the Ericsson's IPTV Mediaroom platform. This data reflects the events and actions taken by the television users and is stored in a central database. Our study does not consider all the possible event types but instead focuses on extracting insights from the most relevant and representative types, like channel or program visualizations. Besides the subscriber events (also called activity logs), that represent the facts in the data warehouse, we also need the infrastructure information to establish the proper context. Extracted from the Mediaroom platform, we have the information that concerns to the infrastructure at a logical level, like the users, their set-top boxes, the television channels and the programs. This information constitutes the infrastructure inventory, and it is modeled in the DW as dimension tables.

4.4.1. Inventory information

The inventory information reflects the different entities that are a part of the Mediaroom platform, and, therefore, it is fundamental to understand their purpose and interconnections so that we can design and represent them, not only accurately, but also efficiently. The entities are defined in a non-enforced relational model (there are no referential integrity constraints), and they are the following:

- **Set-Top Boxes** – Users' home devices that are used to access the television services available on the IPTV platform;
- **Subscribers** – Subscribers of the IPTV service. One subscriber can own multiple set-top boxes (STBs);
- **Services** – The services available on the platform. The most common services are the Live television, the Video-on-Demand (VoD) and the Digital Video Recording (DVR);
- **TV Channels** – Television channels available on the Live service;
- **Service Collections** – a set of Live TV services that together present a consistent content view in display contexts, like the program info screen displayed on the client. An example is a given TV program that is composed of two distinct streams that are presented together, the main program and rolling band at the bottom, called Picture-in-Picture (PiP);
- **Programs** – Complete list of television programs that are transmitted on the Live television channels;

¹⁴ A data lake is a storage repository of vast amounts of raw data that can be either structured, semi-structured or unstructured (Barnes et al., 2016).

- **Subscriber Groups** – A subscriber group consists of a set of services like, for example, the “Sports Package” that contains multiple television channels;
- **Channel Maps** – Channel maps provide a complete list of channels that a subscriber group can access;
- **VoD Assets** – The repository of Video-on-Demand assets that can be visualized by the users.

These entities can, like on any relational model, be connected to each other, and for that purpose, we have a set of relationships that represent these logical connections. They are the following:

- **Mapping between Subscriber Groups and Subscribers** – Subscribers are related to the Subscriber Groups so that it is possible to know to which services they have access to;
- **Mapping between Subscriber Groups and Set-Top Boxes** – Similar to the relationship between Subscriber Groups and Subscribers. It is possible to define, for each individual set-top box, to which services they have access to;
- **Mapping between Service Collections and Services** – This mapping aggregates several and distinct types of services under the entity called Service Collection;
- **Mapping between TV Channels and the platform Channel Mapping** – This relationship instantiates the Channel Maps with the corresponding virtual channel numbers. The Channel Maps are used by the client devices to determine which channels are displayed in the program guide. Simply put, a channel mapping defines that a given channel is being transmitted on a specific channel position (e.g., the “Discovery Channel” is being transmitted on the channel position 130).

4.4.2. Subscriber events

The subscriber events, or also known as activity logs, reflect the events happening on the IPTV platform. These events can happen as a result of the user’s actions, like changing a television channel, or can be simply part of the normal operation of the platform, like the change of program when the previous one finishes and the streaming of a new one begins. The table below identifies the event types that are part of the subset covered by our study.

Event Type	Event Type Name	Description
100	Channel Tune	This event happens when an end user tunes away from a TV channel. This event is only recorded if the user remains tuned to the channel for a specified minimum amount of time (the default defined by the platform is 20 seconds)
101	Box Power	This event reports a change in a set-top box’s power state (on or off)
104	Trick State	This event occurs when a set-top box changes to a new trick state such as fast-forward, play or reverse
114	Program Watched	The Program Watched event is recorded when the program, that an end user is watching, changes to another program

115	DVR Start Recording	A DVR Start Recording event occurs when a set-top box starts recording a program
116	DVR Abort Recording	A DVR Abort Recording event occurs when an ongoing recording is manually stopped before its originally scheduled stop time
117	DVR Playback Recording	A DVR Playback Recording event occurs when the user plays back a recording
118	DVR Schedule Recording	A DVR Schedule Recording event happens when the user schedules a program to record
119	DVR Delete Recording	A DVR Delete Recording event occurs when the user deletes a recording
120	DVR Cancel Recording	A DVR Cancel Recording event occurs when the user cancels a scheduled recording

Table 4.2. Mediaroom event types

The selection of these particular event types is related to their volume and their meaning. These events represent clear user behaviors and are the ones that are generated in greater number.

Detailed information about each of these event types is shown in Appendix A.

4.4.3. Source files

All the source data used in this study is composed of text files that were already extracted from the Mediaroom platform. We have the files that represent the snapshot of the infrastructure and the files containing subscriber events. The fields inside the files are semi-colon separated, and quotes enclose the text fields. In the figure below we present an example of the source files.

```
20160504,"2016-05-04 12:47:43","b4930555-a04f-482a-b3ed-ca496912c0ec","104","7af1141c-96f7-45c9-b589-dacdb8119cae","FastForward","";
20160504,"2016-05-04 12:47:45","b4930555-a04f-482a-b3ed-ca496912c0ec","104","7af1141c-96f7-45c9-b589-dacdb8119cae","Play","";
20160504,"2016-05-04 12:47:47","cfb1cf24-ab53-490c-9a9c-e8777d502243","104","5a956c24-e40e-4b9b-beae-46c9aba139d9","FastForward","";
20160504,"2016-05-04 12:47:47","327b0715-88ea-4239-a86d-8acda9b632a4","104","","Play","";
20160504,"2016-05-04 12:47:48","976886f1-9406-4b2e-b7dc-8c1a071eedf8","104","a9f2727b-b317-4352-84d6-18e75418e31d","Skip","";
20160504,"2016-05-04 12:47:48","7b9f9c60-041c-471b-bb01-d16241e87a2b","MOTOROLA VIP1232E_CE","101","","ON","";
20160504,"2016-05-04 12:47:49","cfb1cf24-ab53-490c-9a9c-e8777d502243","104","5a956c24-e40e-4b9b-beae-46c9aba139d9","FastForward","";
20160504,"2016-05-04 12:47:49","976886f1-9406-4b2e-b7dc-8c1a071eedf8","104","a9f2727b-b317-4352-84d6-18e75418e31d","Skip","";
20160504,"2016-05-04 12:47:51","cfb1cf24-ab53-490c-9a9c-e8777d502243","104","5a956c24-e40e-4b9b-beae-46c9aba139d9","Play","";
20160504,"2016-05-04 12:47:54","9a7a1804-a703-4ae3-b492-d406a32e69f1","MOTOROLA VIP1232E_CE","101","","ON","";
20160504,"2016-05-04 12:47:55","cfb1cf24-ab53-490c-9a9c-e8777d502243","104","5a956c24-e40e-4b9b-beae-46c9aba139d9","Pause","";
20160504,"2016-05-04 12:47:55","d9db888d-a645-4a52-950d-9472dc37f2bb","104","f14187bf-b990-4f9d-9191-601f0a739176","Skip","";
20160504,"2016-05-04 12:47:57","d9db888d-a645-4a52-950d-9472dc37f2bb","104","f14187bf-b990-4f9d-9191-601f0a739176","Replay","";
20160504,"2016-05-04 12:48:00","d9db888d-a645-4a52-950d-9472dc37f2bb","104","f14187bf-b990-4f9d-9191-601f0a739176","Replay","";
20160503,"2016-05-03 19:20:08","cc07c65c-4e60-4fe1-a9ba-3e438032991c","114","008261c2-0000-0000-0000-000000000000","1198","635978941717730000";
20160503,"2016-05-03 19:20:08","ea1214b9-a0e5-4907-b405-646f98db6b5","114","00824b38-0000-0000-0000-000000000000","2393","63597884431620000";
20160503,"2016-05-03 19:20:08","3755e14d-e06d-4b29-8923-f40229250a27","114","005fb06a-0000-0000-0000-000000000000","1276","635978956943360000";
20160503,"2016-05-03 19:20:08","ca66ebc-2f40-45da-941c-e38ea4328bb2","114","008261c2-0000-0000-0000-000000000000","1201","635978952925070000";
20160503,"2016-05-03 19:20:08","683a482d-8a3f-41e6-a898-03c8efdb0244","114","008261c2-0000-0000-0000-000000000000","21","635978930741470000";
20160503,"2016-05-03 19:20:08","0a427663-f6da-41af-a4f6-e5d209264a73","114","008261c2-0000-0000-0000-000000000000","1198","635978548299750000";
20160503,"2016-05-03 19:20:08","7cece350-ac24-4dab-98fe-516d2c8915c","114","005fb06a-0000-0000-0000-000000000000","256","63597884431620000";
20160503,"2016-05-03 19:20:08","0eaf8f02-d5ad-44df-bf7a-8b3f80288226","114","00824b38-0000-0000-0000-000000000000","6599","635978896071260000";
20160503,"2016-05-03 19:20:08","55e99f0f-6c00-44d7-959a-29aca5c88e5b","114","00824aef-0000-0000-0000-000000000000","5312","63597896063770000";
20160503,"2016-05-03 19:20:09","784841b0-868a-46d4-b06e-9bfdd7157e04","114","008261c2-0000-0000-0000-000000000000","1201","63597889725190000";
20160503,"2016-05-03 19:20:09","4a07f558-5d54-404d-af5a-72d98b059b72","114","00824b38-0000-0000-0000-000000000000","1683","63597895025250000";
20160503,"2016-05-03 19:20:09","be676208-cadd-45e6-b05f-208cdcab148c","114","008261c2-0000-0000-0000-000000000000","484","635978963588910000";
20160503,"2016-05-03 19:20:09","d5ef3cdc-6f44-4d5a-9268-fe62f202848f","114","00824b38-0000-0000-0000-000000000000","6600","63597895547570000";
20160503,"2016-05-03 19:20:09","30788847-05a5-43b0-96db-60b79e1c84cf","114","005fb07a-0000-0000-0000-000000000000","412","635978961803570000";
20160503,"2016-05-03 19:20:09","39a0839e-4350-4082-85c4-789c80bb5295","114","00824b38-0000-0000-0000-000000000000","6600","63597894592620000";
20160503,"2016-05-03 19:20:09","80190c43-59ec-44d0-8c67-476938f008e3","114","005fb06a-0000-0000-0000-000000000000","1380","635978879750640000";
20160503,"2016-05-03 19:20:09","69259cc1-6f00-43a4-a741-fe6f370bc861","114","005fb06a-0000-0000-0000-000000000000","1380","63597889088610000";
20160503,"2016-05-03 19:20:09","c116058c-ec44-4b71-8a3e-4257bc39c4c4","114","005fb07a-0000-0000-0000-000000000000","1379","635978960915340000";
20160503,"2016-05-03 19:20:09","33ca7a81-674d-44de-b1bb-b377f407f80","114","005fb06a-0000-0000-0000-000000000000","1378","635978944409180000";
20160503,"2016-05-03 19:20:09","aaf7bb07b-9801-4bd1-8412-8a9d7bd67620","114","008261c2-0000-0000-0000-000000000000","225","635978953584780000";
20160503,"2016-05-03 19:20:09","1e758bb-2937-4580-beaa-8a30f564aef","114","00824b38-0000-0000-0000-000000000000","2598","635978207190650000";
```

Figure 4.3. Source file sample

During the evaluation phase of our research, we compare and benchmark the transformation processes designed in both the DW and the Hadoop cluster, and to help us achieve that, in a more controlled and systematic approach, the source files, regarding the subscriber events, were specifically prepared for the task.

4.4.4. Data preparation

Multiple days and multiple event types are present in a single subscriber events file. This is the norm since the events are sent without any particular order, to the central repository, by each set-top box when its buffer reaches the limit. These heterogeneous source files were prepared with the purpose of creating similar files in size, split by day and event type.

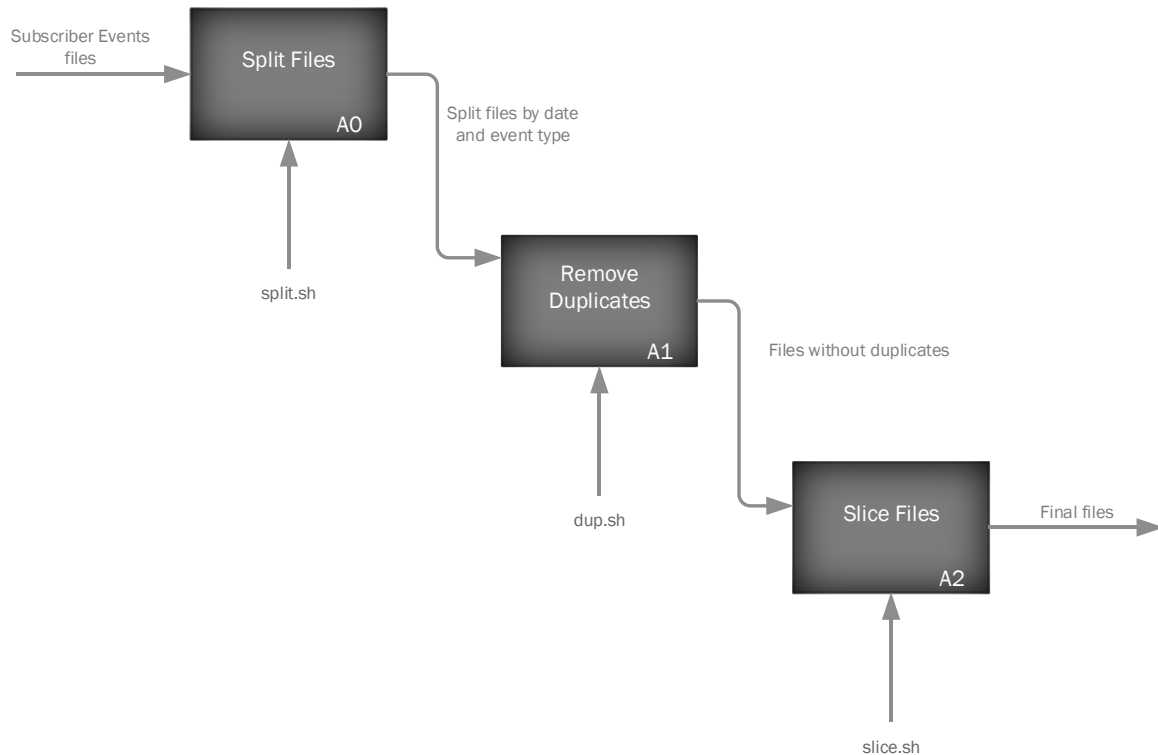


Figure 4.4. Data preparation functional model

The subscriber events files were processed by the three procedures shown in the diagram above.

- **A0: Split Files** – the source files are decompressed and then split, into multiple files, as per their content. A file is generated for each combination of date (YYYYMMDD) and event type;
- **A1: Remove Duplicates** – the files generated in the previous procedure are analyzed, and any duplicate record is removed. The generation of the output files is compressed on the fly and the source file removed with the purpose of saving space in the file system;
- **A2: Slice Files** – the files, already without duplicates, are firstly decompressed and then sliced into 200MB files. The final files generated by this procedure are then compressed back again since all the source files, for both the RDBMS and the Hadoop cluster, will be in a compressed format.

The subscriber event files, generated by this process, follow the naming convention 'MR_AL_YYYYMMDD_nnn.txt.gz' where:

- MR – Mediaroom;
- AL – Activity Logs;
- YYYYMMDD – Day identifier;
- nnn – Event type identifier.

These files have the extension 'txt.gz' since they are compressed text files. An example of a file name produced after the data preparation is: MR_AL_20160601_100.txt.gz. The same exact files are used to feed both environments, the RDBMS and the Hadoop cluster.

The scripts used on this data preparation step can be found in Appendix B.

4.5. DATA WAREHOUSE DESIGN

The data that is extracted from the IPTV platform, in its raw form, needs to be transformed into meaningful information so that it can be afterward made available in the data warehouse. The transformation stage is performed inside the DW itself, meaning that we are working with an Extract-Load-Transform approach.

The data warehouse is composed of a Staging Area where both the inventory and the subscriber events, extracted from the Mediaroom, are initially loaded. The transformation processes then use this data to populate, according to a set of predefined rules, the definitive inventory and fact tables at the core of the DW. Afterward, the events are used to produce the aggregations that will ultimately feed the reporting layer. This flow and its related processes are presented in Figure 4.5.

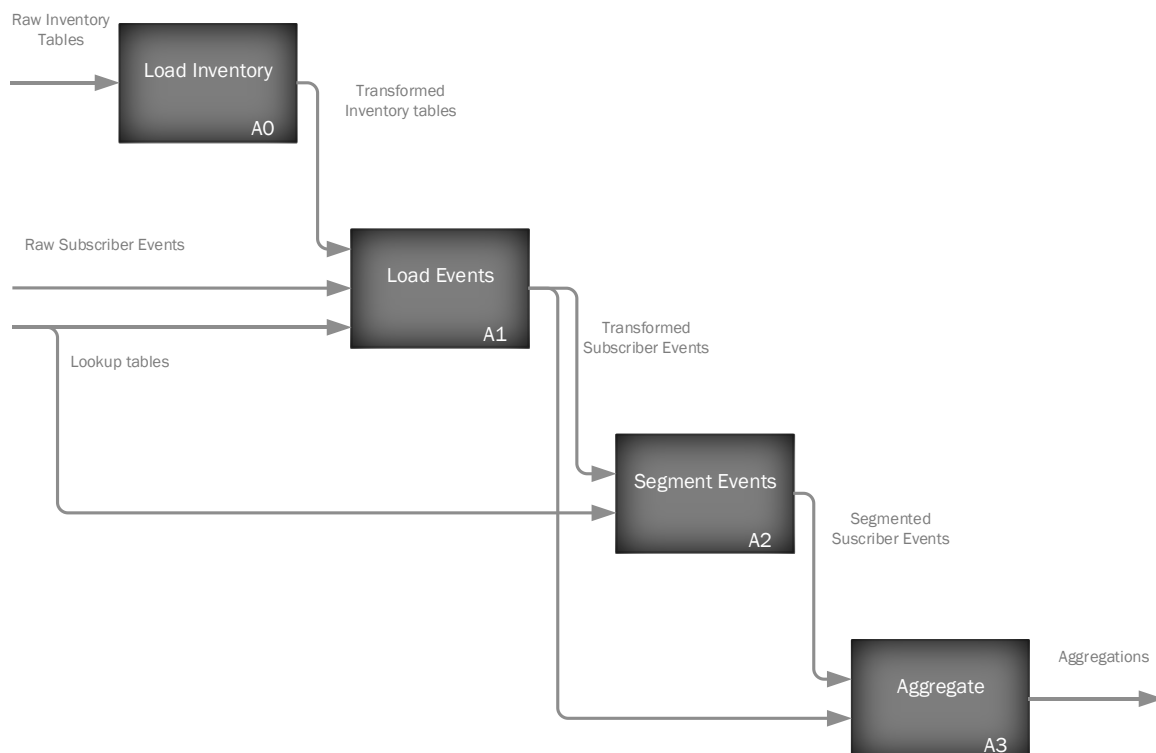


Figure 4.5. Data warehouse functional model

Between the 'Load Events' (A1) and the 'Aggregate' (A3) we have an extra activity that segments the events into five-minute slots to produce the heaviest aggregation delivered by the system, the television audience measurements for every five-minute interval during the entire day.

The source data needs to be transformed into valid and meaningful information for the consumers of the system. This study focused on the diversity of the transformations rather than in the complete coverage of the required transformation procedures. We are driven by the ultimate goal that is producing the valuable insights from the input data, but in this study, we are focusing on the processes themselves so that we can gather information and reach conclusions regarding the usefulness of a Hadoop cluster as a transformation tool in a traditional Data Warehousing context. With this in mind, in this chapter, we cover a set of distinct transformation processes that range from the initial data loading to the last phase, the analytical part where the information is aggregated, and the metrics produced.

From the analysis of the IPTV infrastructure and the data we can extract from it, we identified a set of entities and their associated relationships, that put together in a data model, are capable of expressing effectively the pertinent information concerning television audience measurements. This data model follows the principles of dimensional modeling where the relevant measurements are expressed as facts, and the entities that give them context are defined as dimensions. However, before the data can be integrated into the core of the data warehouse, it needs to undergo a series of transformation steps to assure several of the dimensions of data quality like conformity, completeness or integrity. All the required tables to the construction of the data warehouse are identified and described in the next five sections, and in the sixth section, we identify the main transformation processes that are subject to our evaluation tests later in the research. A detailed data dictionary of each of the tables in the data model is described in Appendix C.

4.5.1. Staging Area tables

These tables, listed on Table 4.3, map the source files directly and they are responsible for feeding the data into the data warehouse.

Table Name	Description
SA_ACTIVITY_EVENTS	Activity logs representing the user behaviors
SA_ASSET	Inventory of VoD assets
SA_CHANNEL_MAP	Channel maps list
SA_GROUP	List of subscriber groups
SA_PROGRAM	Inventory of television programs
SA_SERVICE	List of services
SA_SERVICE_COLLECTION	List of service collections
SA_SERVICE_COLLECTION_MAP	Mapping between the service collections and the services they contain
SA_STB	List of set-top boxes
SA_SUBSCRIBER_GROUP_MAP	Mapping between the subscribers and the groups to which they are assigned to
SA_TV_CHANNEL	List of television channels

Table 4.3. Staging Area tables

4.5.2. Inventory tables

The Inventory tables, listed on Table 4.4, are responsible for describing the Mediaroom platform inside the DW. They identify and characterize its entities, relationships, and configurations. These tables are, in most cases, an improved mirror of the staging tables but they also describe, in a more explicit way, certain relationships with the purpose of facilitating the transformation of the subscriber events.

Table Name	Description
SS_ASSET	Inventory of VoD assets
SS_CHANNEL_MAP	Channel maps list
SS_GROUP	List of subscriber groups
SS_MAP_CHANNEL_MAP_SERVICE	Mapping between the channel maps and their corresponding services
SS_MAP_STB_CHANNEL_MAP	Mapping between the set-top boxes and their assigned channel maps
SS_PROGRAM	Inventory of television programs
SS_SERVICE	List of services
SS_SERVICE_COLLECTION	List of service collections
SS_SERVICE_COLLECTION_MAP	Mapping between the service collections and the services
SS_STB	List of set-top boxes
SS_STB_GROUP_MAP	Mapping between the set-top boxes and the groups to which they are assigned to
SS_SUBSCRIBER_GROUP_MAP	Mapping between the subscribers and the groups to which they are assigned to
SS_TV_CHANNEL	List of television channels

Table 4.4. Inventory tables

4.5.3. Support tables

The support tables are used as straightforward dimensions, inside the DW, with the purpose of facilitating the representation of time. The only two support tables are presented below.

Table Name	Description
LU_DATE	List of days in the 'YYYYMMDD' format
LU_START_GP	List of granularity periods, i.e., slices of time, in a day, according to a specified duration (5, 15, 30 or 60 minutes)

Table 4.5. Support tables

4.5.4. Fact tables

In our data warehouse we have one fact table that stores all the different types of television viewing behaviors. However, two particular type of events, the *Channel Tune* and the *Program Watched*, are segmented in five-minute slots, and this result is stored temporarily in a second fact table that is used exclusively as a helper for the aggregations. Both these fact tables are described on Table 4.6.

Table Name	Description
FACT_ACTIVITY_EVENTS	Fact table containing the activity logs that represent the user behaviors while using the IPTV service
FACT_EVT_SEGMENTED	Fact table containing the <i>Channel Tune</i> or the <i>Program Watched</i> events, segmented by five-minute periods. This table is used as source for the aggregation processes that calculate the audience measurements

Table 4.6. Fact tables

4.5.5. Aggregation tables

The aggregation tables, listed on Table 4.7, are the ultimate target of the transformation processes, as they store the calculated metrics that are then presented by the reporting layer.

Table Name	Description
AG_LIVE_SHARE_GP	Aggregation table reporting the television channel audience by five-minute periods per day
AG_LIVE_RATING_DY	Aggregation table reporting the daily program rating
AG_LIVE_REACH_DY	Aggregation table reporting the daily channel reach

Table 4.7. Aggregation tables

4.5.6. Transformation processes

To better understand the transformation and aggregation processes, it is important to categorize them according to their input, output, the volume of data and of course how the data is processed. Through this distinction, we can better understand where performance advantages can be gained and with this information, we are in a position to determine which processes are the best candidates to be moved out from the RDBMS and into the Hadoop cluster, to maximize the overall efficiency of the data warehouse.

We present the Data Flow Diagrams (DFDs) that capture the most relevant processes inside the DW and that were benchmarked in both systems, the RDBMS and the Hadoop cluster. These DFDs are however colored in a way so that we can visually and easily understand the volumes of data being used and produced. Table 4.8 identifies and describes the used color scheme.





Color	Table Type	Size
	Small table	Less than 50.000 records
	Medium table	Between 50.000 and 2.000.000 records
	Big table	Between 2.000.000 and 20.000.000 records
	Huge table	More than 20.000.0000 records

Table 4.8. Color scheme for the data volume representation

4.5.6.1. Subscriber events transformation

Different event types have different transformation rules since they represent very distinct concepts. We have events reporting channel changes, program changes, DVR actions or simple set-top box power on/power off actions. These different behaviors also represent very dissimilar volumes of data, and of course, it is important to discern their execution and performance.

When we analyze the subscriber events transformation, it is important to point out that these processes are purely transformation processes, in the sense that the number of records at the input is exactly the same as the one at the output. The transformation rules are responsible for enriching these events in the data warehouse context so that afterward we can extract the desired insights.

The transformation of the *Channel Tune* event (Event Type 100), depicted in Figure 4.6, uses as source the input files extracted from the Mediaroom platform and adds to these records extra information like the television channel that they are reporting. This is only possible by joining the events with the mappings between the STBs and their associated Channel Maps that relate them to the TV channels.

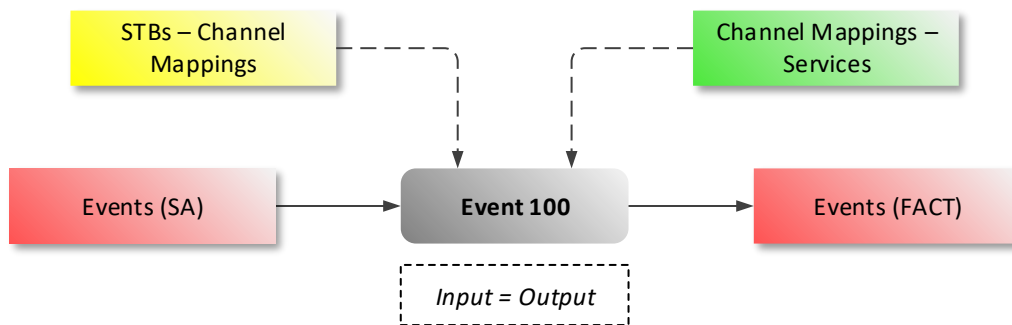


Figure 4.6. *Channel Tune* event transformation DFD

This process can transform more than ten million records that are enriched through a *join* with a medium size table, containing more than a million records, and also with a small table populated with less than one thousand records.

The *Program Watched* event (Event Type 114), depicted in Figure 4.7, follows the same principle of event *Channel Tune* but adds an extra complexity layer due to the amount of data used in its transformation. To identify to which channel the program being watched belongs, we need to perform a *join* between the raw *Program Watched* events and the already transformed *Channel Tune* events.

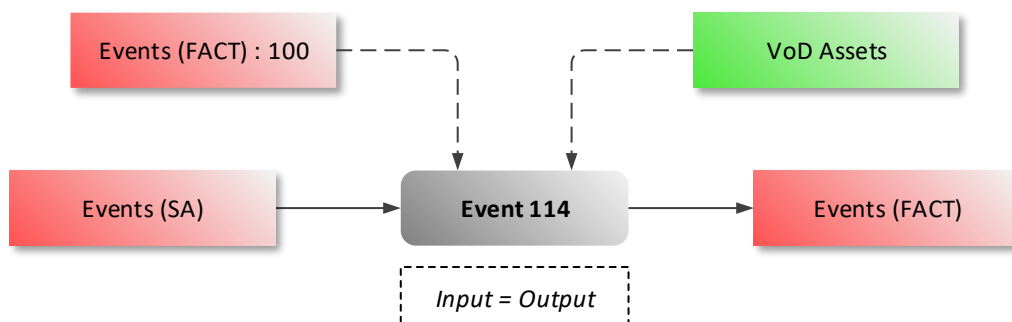


Figure 4.7. *Program Watched* event transformation DFD

In this case, we are joining two big sets of data containing, each more than ten million records. A third table is also used, containing the information about the VoDs, but the size of this table is minimal.

Volume-wise, the process in Figure 4.8, is a light transformation since the subscriber events reporting Digital Video Recordings (Event Types from 115 to 120) are not so frequent when compared with other event types. It is normal that the number of programs being recorded is significantly smaller than the number of, for example, programs being watched. Unlike the transformations of *Channel Tune* and *Program Watched* events, here the volume is around just one tenth of them, approximately one million.

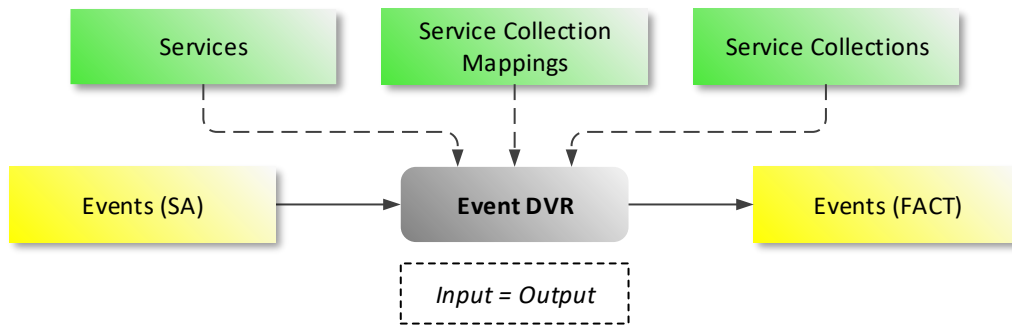


Figure 4.8. DVR Events transformation DFD

This transformation, on the other hand, is using multiple event types and performing different transformations on the same iteration, this according to the particularity of each event type.

4.5.6.2. Subscriber events segmentation

The *Event Segmentation* purpose is to facilitate the aggregation that will report viewing measurements in five-minute intervals. *Channel Tune* and *Program Watched* events are facts that have a start time and a duration, and from the combination of these two attributes, we can place them in a time interval.

Knowing which users or set-top boxes were tuned in a given channel and in a given five-minute interval, from millions of records, requires a carefully designed process. The method that was followed took a phased approach where firstly we create segments of the events for each five-minute slot. The result has as many segments, as many five-minute slots are crossed by the event since its beginning until its end, having in consideration the start time and the duration.

This process has the particularity that it multiplies the number of records used as input by their duration, or more precisely, by the number of five-minute slots it traverses.

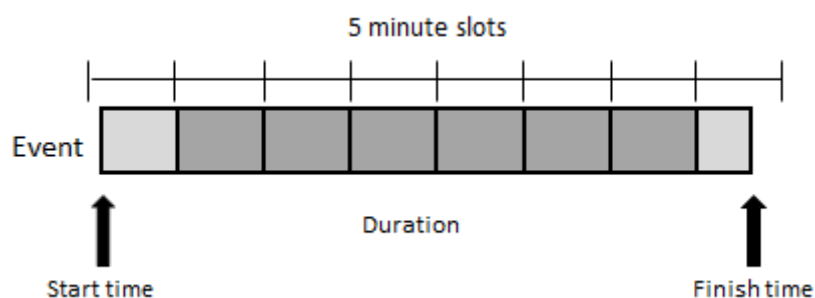


Figure 4.9. Event Segmentation example

In Figure 4.9, we have an event traversing eight different five-minute slots, even though its beginning and ending do not cross the entire slot. Either way, this event, through the segmentation process, will originate eight different records and these records will then be used as input for the aggregation process in charge of creating the five-minute television viewing metrics.

The challenge here is once again volume but in a different perspective; this process, illustrated in Figure 4.10, is responsible for the creation of new facts that increase the level of granularity and the volume of data. From an already big input of about ten million records, we can generate an output multiple times bigger (approximately seven times bigger).

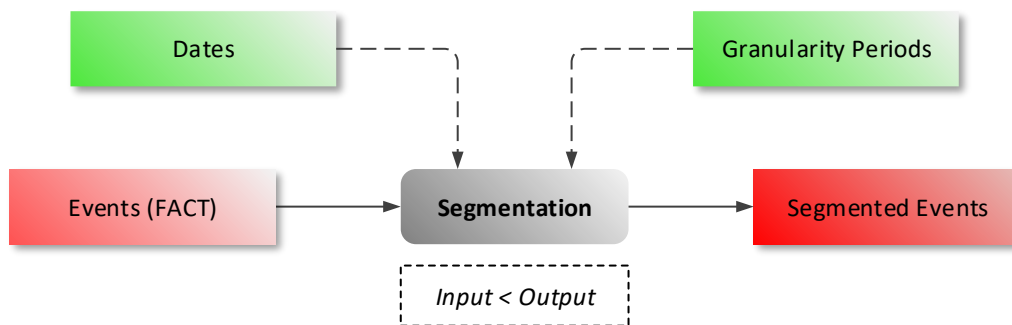


Figure 4.10. Event Segmentation DFD

4.5.6.3. Subscriber events aggregation

Unlike the previous processes, this is not a transformation process, but instead, an aggregation one that generates a small analytical result from a huge input, containing millions of facts.

Aggregation processes are not part of the transformation stage in data warehouses, but many times can represent interesting challenges motivated by the complexity of their calculations or by the amount of data they have to rely upon. In this specific case, the process depicted in Figure 4.11, we are using a huge amount of information, the segmented events, and calculating television viewing metrics that represent an amount of information more than one thousand times smaller than its input.

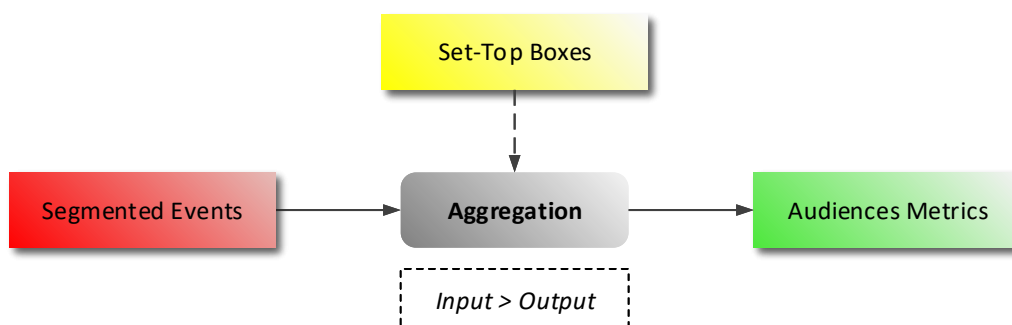


Figure 4.11. Audiences Aggregation DFD

Through this process classification, we are able to represent different processes, mainly discerned by the differences of input size versus output size and, of course, the volumes of data involved. This concludes the design phase of our research, and from this point forward we move to the implementation stage that creates the transformation processes and associated data model, in both the RDBMS and the Hadoop cluster.

4.6. ENVIRONMENT

Before the implementation of the model and processes previously identified, we need to address the questions related to the supporting infrastructure. We need several computer systems capable of hosting the different components participating as support for the implementation and testing of the processes subject to our research. We have the system that represents the data warehouse, the system that acts as the Hadoop cluster, and finally, the system that plays the role of the “Presentation Layer” where the measurements, produced from the raw data, are made available to the users. Additionally, we also need to consider the systems that take part in the scalability tests. The various and diverse systems were installed and configured over a virtual environment implemented by Oracle’s VirtualBox (version 5.1.18). Hosting our virtual environment, we have one single personal computer with the hardware specifications described in Table 4.9.

Component	Specifications
Processor	Intel Core i7-4770S, 3100 MHz (QuadCore)
Motherboard	Asus Z87-Pro (4 PCI-E x1, 3 PCI-E x16, 4 DDR3 DIMM, Gigabit LAN)
Memory	32 GB (4 x Kingston HyperX KHX1866C9D3/8GX)
Graphic card	MSI NVIDIA GeForce GTX 750 Ti (2 GB)
Storage	Samsung SSD 840 EVO 120GB (120 GB, SATA-III)
	WDC WD10EZEX-00BN5A0 (1000 GB, 7200 RPM, SATA-III)
	WDC WD30EZRX-00SPEB0 (3000 GB, 5400 RPM, SATA-III)
	WDC WD6400AAKS-65A7B0 (640 GB, 7200 RPM, SATA-II)
	SanDisk Ultra (128 GB USB3)
	Seagate ST332062 0AS USB Device (320 GB, 7200 RPM, SATA-II)
Network	Seagate ST332062 0AS USB Device (320 GB, 7200 RPM, SATA-II)
	Intel Ethernet Connection I217-V
Operating System	Windows 10 Professional 64-bit

Table 4.9. Hardware specifications of the host computer

4.7. RDBMS IMPLEMENTATION

The Relational Database Management System used to support the traditional data warehouse, central in this study, is built on top of an Oracle database. Oracle databases have been around since 1980, and they still are the most widely used RDBMS nowadays. Their robustness, performance, and features makes them one of the top performers in the RDBMS world (DB-Engines, 2017).

4.7.1. RDBMS infrastructure

As mentioned, the database system supporting our data warehouse, is an Oracle, more specifically, the Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 – 64bit Production with the Partitioning, OLAP, Advanced Analytics and Real Application Testing options. The database is installed on top of an Oracle Linux Server 7.2 with Unbreakable Enterprise Kernel (3.8.13-118.13.2.el7uek.x86_64) and has the hardware specifications described in Table 4.10.

OS	RDBMS	Hardware (virtual)
Oracle Linux Server 7.2	Oracle Database 12c Enterprise Edition	CPU: 2 cores Memory: 8 GB (Maximum memory target: 5 GB) Disk: WDC WD30EZR (5400 RPM SATA-III)

Table 4.10. Data warehouse environment

Details about the installation of the Oracle database for this study can be found in Annex A. This chapter was built as annex so that it can also be used independently from our research, as a step-by-step guide on how to install a database instance in a virtual environment using Oracle VirtualBox.

4.7.2. RDBMS configuration

The configuration of the RDBMS was minimal with the purpose of making use of many of the automatic management mechanisms offered by Oracle DBMS, like the Automatic Memory Management that balances memory allocation between the System Global Area (SGA), and the Program Global Area (PGA) according to the database's usage. In Figure 4.12 we can observe the memory configuration that sets the maximum target to 5 GB. Additional configuration parameters were set during the database installation and can be confirmed in Annex A.

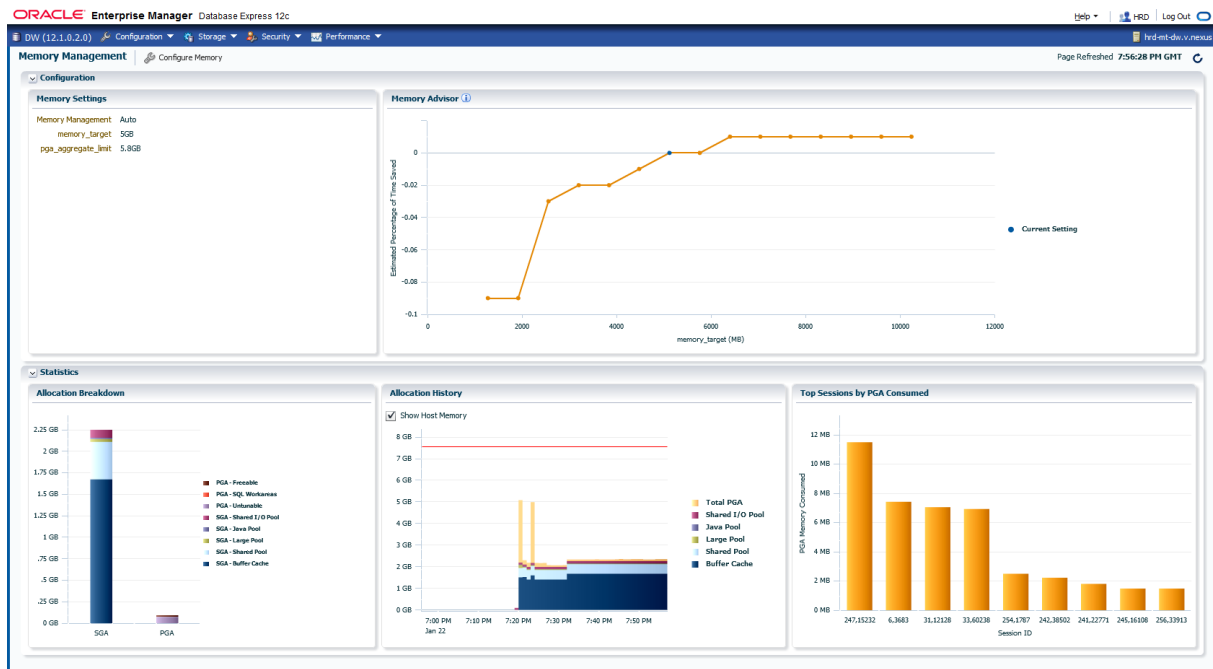


Figure 4.12. RDBMS memory configuration

Still part of the database configuration, and part of Oracle’s architecture, we need to create the tablespaces where the tables are being stored, and configure the system tablespaces used by the RDBMS to perform its operations. The table below shows us the tablespace configuration and its DDL is in Appendix D.2.

Tablespace	Size	Description
HRD_DW_AGG	1 GB	Tablespace for the aggregation tables
HRD_DW_DAT	12 GB	Tablespace for the fact table containing the subscriber events
HRD_DW_INV	2 GB	Tablespace for the inventory tables
HRD_DW_SEG	8 GB	Tablespace exclusively for the segmented table
TEMP	24 GB	System tablespace used for actions like <i>hash</i> or <i>merge joins</i> when they do not fit in memory. In our study, we have big <i>hash joins</i> that require a large temporary tablespace
UNDOTBS1	1 GB	System tablespace used for rollback operations. In our study, we perform mostly insert operations in direct-load mode, and because of that the usage of this tablespace is minimal
SYSTEM	1 GB	System tablespace containing the information about the structure and contents of the database
SYSaux	1 GB	Secondary system tablespace containing the information about the structure and contents of the database

Table 4.11. Tablespace configuration

There are no specific tablespaces for the indexes since their usage is reduced to the minimum, especially because we are mostly performing big operations like *merge* or *hash joins* that use entire tables or entire partitions. Adding indexes in this scenario would give us no benefit, and it would damage performance for the insert operations.

The configuration of this database, for the purposes of our study, represents the limit of vertical scalability of the data warehouse.

4.7.3. Data model design

In this section, we address the design and creation of the multiple tables implementing the data warehouse's data model on all its components – the Staging Area, the Fact and associated Dimension tables, and finally, the Aggregation tables where the calculated metrics are stored.

4.7.3.1. Staging Area tables

The Staging Area tables store the source data, composed of the files extracted from the Mediaroom platform and compressed to save space in the file system. The diagram in Figure 4.13 illustrates all the Staging Area tables that were created as Oracle external tables and its DDL is presented in Appendix D.4. Before the creation of these external tables, directories as Oracle objects were created with the DDL shown in Appendix D.3, so that the files could be mapped directly from the file system into the database.

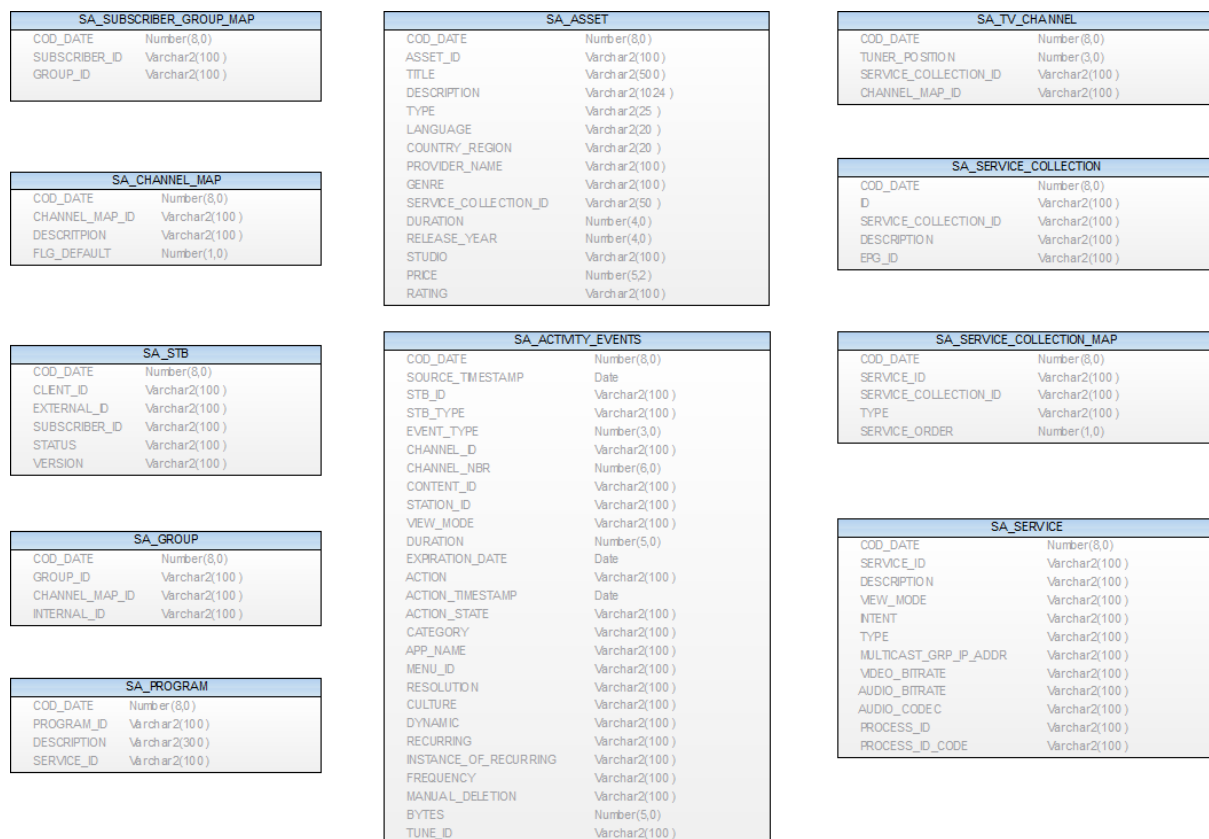


Figure 4.13. Staging Area tables physical model in the RDBMS

4.7.3.2. Inventory tables

The Inventory tables reflect the Mediaroom infrastructure on a daily basis. Under this design, the tables are partitioned by day, where each day represents a snapshot of the Mediaroom entities identified and described in section 4.4.1. These Inventory tables have a dual function in our data warehouse. Part of them act as dimension tables, which contextualize the facts, and others are used simply as support for the transformation processes.

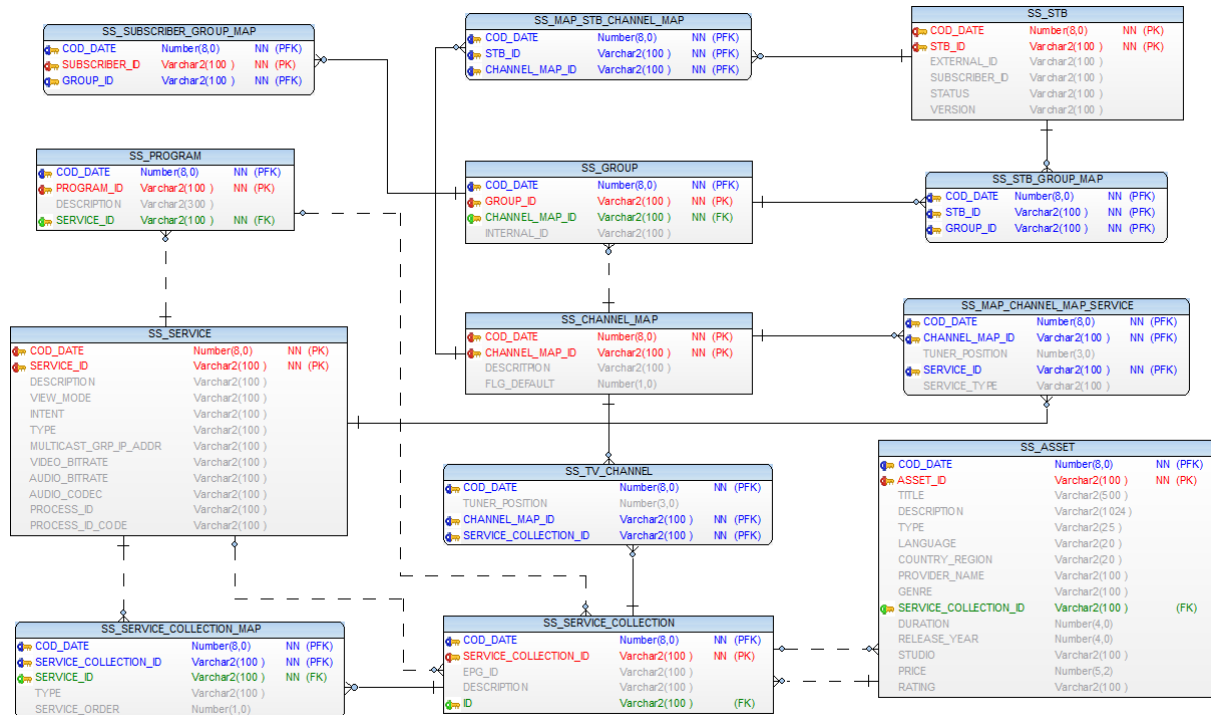


Figure 4.14. Inventory tables physical model in the RDBMS

The relationships depicted in the diagram above are merely informative since they are not enabled in the physical model. All the relationships, rather than being explicit, are implicit and represented through the column names. Having an enabled referential integrity definition in a data warehouse has performance costs (Ordenez & García-García, 2008) and its implementation brings some extra complexity especially when dealing with partitioned tables (Imhoff et al., 2003; Kimball & Ross, 2013).

The DDL and the DML statements used to create and load the Inventory tables are available in Appendix D.6.

4.7.3.3. Fact tables

The fact tables are at the core of every data warehouse as they capture, at a granular level, the measurements being studied. Our fact tables store the different events extracted from Mediaroom. As these events represent different user behaviors, with different characteristics and volumes, our fact tables are partitioned by day and sub-partitioned by event type with the purpose of optimizing data access performance.

Fact tables also store very large amounts of data and consequently are heavy consumers of the storage resources. To find the right balance between storage consumption and performance, we conducted a series of tests, described in Appendix D.1, and concluded that the best approach, from the possible options, would be to store the fact tables using the *row store basic* compression format.

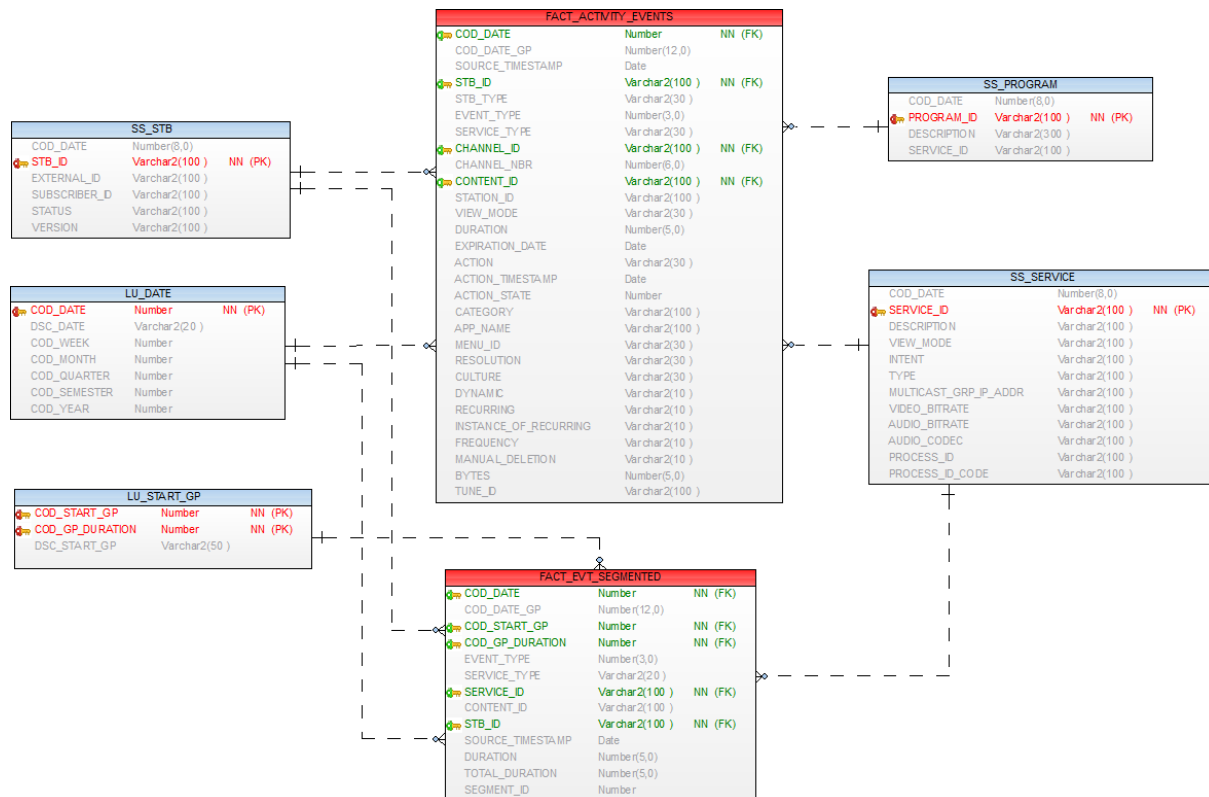


Figure 4.15. Fact tables physical model in the RDBMS

As for the Inventory tables, also here the relationships, depicted in Figure 4.15, are informative since they are not enabled in the data model. The purpose of our fact tables is described in section 4.5.4, and its DDL and DML statements are in Appendix D.7.

4.7.3.4. Aggregation tables

Aggregations store summarized analyses of the facts and are normally accessed by the Business Intelligence layer. The design and implementation of aggregation tables is tightly connected to this layer as it defines the information requirements that need to be answered by the data warehouse. For our research, we defined three aggregation tables, described in section 4.5.5, that can deliver the previously defined audience measurements *Rating*, *Reach* and *Share*. These aggregations are also some of the ones that rely on large amounts of data and, therefore, perfect candidates for the benchmarking tests performed by our study.

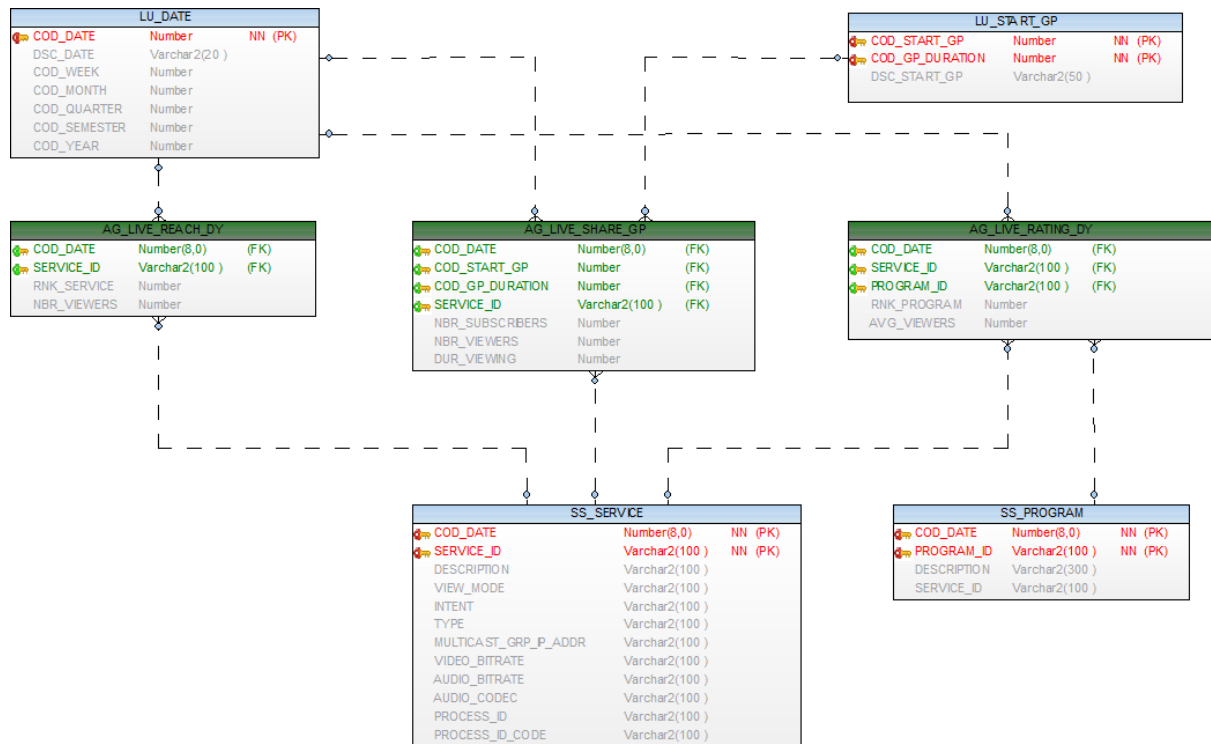


Figure 4.16. Aggregation tables physical model in the RDBMS

The aggregation tables presented above, store small datasets and therefore represent no concern regarding performance. Nevertheless, they are, like the fact tables, partitioned by date (their DDL, and related DML, is available in Appendix D.8). This facilitates aging mechanisms and also the processes in place with the purpose of exchanging data between the Hadoop cluster and the RDBMS. The depicted relationships, between the aggregation and dimension tables serve, also here, the purpose of contextualizing the information since they are not enabled in the physical model.

4.7.4. Data transformation

The implementation of the processes is a fundamental step in our research as they are responsible for the transformation of the raw data into meaningful measurements. These processes were identified and described in section 4.5.6, and at this stage, they are materialized in SQL statements. For our study, rather than exhaustively analyze all the required processes, we focused on a subset that, according to their intrinsic characteristics, allow us to infer the conclusions of their testing onto other similar processes. To calculate the proposed audience measurements, other processes were implemented, but throughout the benchmarking tests, the processes identified in Table 4.12 were our focus.

Process	Type ¹⁵	Description
Channel Tune	1:1	Transformation process that takes one input record and also generates one record as output (uses medium sized <i>joins</i>)
Program Watched	1:1	Transformation process that takes one input record and also generates one record as output (uses a large sized <i>join</i>)

¹⁵ '1:1' stands for *One-to-One*, '1:M' for *One-to-Many* and 'M:1' for *Many-to-One*.

DVR Events	1:1	Transformation process that takes one input record and also generates one record as output (uses multiple small sized <i>joins</i>)
Event Segmentation	1:M	Multiplies each input record into one or more output records. The average ratio of this multiplication is around 1 to 7
Audiences Aggregation	M:1	Aggregates large number of records to produce a single output record and performs analytical operations as well as roll-ups

Table 4.12. Implemented transformation processes

The SQL statements associated with all the processes implemented during our study are disclosed in Appendix D. The statements directly associated with the transformation and aggregation of the subscriber events are detailed in Appendix D.7 and Appendix D.8, respectively.

During the implementation of the SQL statements, our prime guideline was performance, and to that effect, special attention was paid to the execution plans of each of the statements, where execution time was given priority over the execution cost. The transformation statements were implemented with multiple approaches, and the most performant ones were selected. We present the execution plans for the transformation statements in Appendix D.10. We can observe that the statements make the correct use of the physical model, by using its partitioning scheme and that the *joins* are performed via *merge* or *hash joins* and never through *nested loops* where each record is evaluated individually on both sides of the *join* and thus only advisable for small data sets and preferably with indexes (Oracle Corporation, 2017). All our tables were kept with current statistics at all times so that Oracle's Cost Based Optimizer could always base its decisions with the most accurate information regarding the tables' volumes and contents.

Finally, one important aspect is that during the implementation of these transformations in Oracle, we always tried to make use of what is most efficient in Oracle, like the use of helper functions that store their values in cache, without any concerns whatsoever of how this would or could be implemented in Hive. Assessing the number and level of adaptations to the SQL, required to move it from Oracle to Hive, was also an important aspect to gather.

4.8. HADOOP CLUSTER IMPLEMENTATION

Before installing and configuring the Hadoop cluster, supporting the transformation processes being implemented in this study, we analyzed the several options around the Big Data landscape and in particular Hadoop. To build the required Hadoop cluster, we need several components, namely to manage the storage and also to assure the distributed processing management and the associated resource allocation. On top of this, there is also the need for a high-level framework/engine, with the purpose of enabling the most effortless migration of the SQL statements, responsible for the transformation mechanisms inside the RDBMS, that are part of the initial data warehouse architecture.

Every Apache project, like Apache Hadoop, Apache Tez or Apache Hive can be downloaded, installed and configured manually with the purpose of creating a Hadoop cluster. This is a time-consuming task that can be easily avoided by using one of the Hadoop distributions developed by Cloudera,

Hortonworks or MapR. Each of the solutions, offered by these companies, package together several applications that can be easily installed and managed throughout a Hadoop cluster, thus saving us the process of installing and configuring each application in each node. Any of the distributions offer the standard Apache applications, available in Hadoop’s ecosystem, but there are also many differences between them that may or may not have implications in the installation of a cluster. Our study does not include a comprehensive analysis of the features available in each of the distributions, but instead, our approach is rather pragmatic when deciding which of the distributions is better suited.

The first important decision relates to the selection of the SQL-like engine that would be used and here our choice fell onto Hive simply because, according to DB-Engines, Hive ranked, in April of 2016, on the tenth position while Impala appeared only on the twenty-third in the ranking of relational DBMSs (DB-Engines, 2017). By selecting Hive as the relational engine to be used in the Hadoop cluster, Cloudera was immediately discarded as the distribution that would support our study because Cloudera favors Impala as a SQL-like engine in Hadoop and puts most of its efforts in its development. On the other hand, Hortonworks favors Hive and its development, namely through the “Stinger Initiative” (Gates, 2013). When compared to MapR Converged Platform 5.2, Hortonworks Data Platform 2.5 expands the delivered 1.2 Hive version with several features from Hive 2.1. At this point the decision was clear, Hive as the SQL-like engine and the Hadoop cluster installed and managed by Hortonworks Data Platform 2.5.

Another important aspect that we took into consideration, when choosing between Cloudera, Hortonworks or MapR, was that Hortonworks is completely open-source while, for example, MapR uses several proprietary components. Cloudera also has an enhanced commercial license while Hortonworks works under an open-source license with no extra features or costs associated with a commercial license.

4.8.1. Hadoop cluster infrastructure

By making use of our virtual environment, described in section 4.6, we created three virtual machines with the purpose of hosting the cluster’s nodes. For the installation of our nodes, we chose the same operating system like the one used for the RDBMS, the Oracle Linux Server 7.2 with Unbreakable Enterprise Kernel (3.8.13-118.13.2.el7uek.x86_64). The hardware reserved for each of the nodes is detailed in Table 4.13.

Node	OS	Hadoop	Hardware (virtual)
1	Oracle Linux Server 7.2	Hortonworks Data Platform 2.5.3	CPU: 2 cores Memory: 10 GB (7.5 GB for YARN containers) Disk: WDC WD10EZEX (7200 RPM, SATA-III)
2			CPU: 2 cores Memory: 6 GB (5 GB for YARN containers) Disk: WDC WD6400AAKS (7200 RPM, SATA-II)
3			CPU: 1 core Memory: 6 GB (5 GB for YARN containers) Disk: WDC WD30EZR (5400 RPM SATA-III)

Table 4.13. Hadoop cluster environment

The installation of the cluster was done using Hortonworks Data Platform (HDP) 2.5.0¹⁶ and it is detailed in Annex B. This chapter was built as annex so that it can also be used independently from our research, as a step-by-step guide on how to install a Hadoop cluster in a virtual environment using the Ambari managed environment of HDP.

4.8.2. Hadoop cluster configuration

The installation of the cluster through Ambari is a very straightforward task and assures us an initial effective configuration based on default values. Due to the limited resources of our environment, and with the cluster already up and running, we performed several tests to assess the best role and service distribution amongst our three the nodes. The final distribution is presented in Table 4.14.

Node	Component	Services
hrd-mt-h01	HDFS	NameNode DataNode
	YARN	ResourceManager NodeManager
	Hive	Hive Metastore HiveServer2 MySQL Server WebHCat Server
	Others	Knox Gateway (Knox) ZooKeeper Server (ZooKeeper) Metrics Monitor (Ambari Metrics)
hrd-mt-h02	HDFS	SNameNode DataNode
	YARN	App Timeline Server NodeManager
	Others	ZooKeeper Server (ZooKeeper) Metrics Collector (Ambari Metrics) Metrics Monitor (Ambari Metrics)
hrd-mt-h03	HDFS	DataNode
	YARN	NodeManager
	Others	Oozie Server (Oozie) ZooKeeper Server (ZooKeeper)
		History Server (MapReduce2) Infra Solr Instance (Ambari Infra) Metrics Monitor (Ambari Metrics) Grafana (Ambari Metrics)

Table 4.14. Hadoop cluster service distribution

Not all the services of the table above are required or, at least, they do not need to be always running, for our Hive on Tez to be able to execute our data transformation processes. The basis of the cluster is its data storage implemented by HDFS, and to that effect, each node is a DataNode. The central part of HDFS is the NameNode, where the metadata regarding all the data distributed across the cluster is stored. Since Hadoop 2, this critical role is safeguarded by a Secondary NameNode that can step in to

¹⁶ Later the HDP version was upgraded to 2.5.3.

cover for a potential failure of the NameNode. We recognize that it is not the best practice to have the roles of DataNode and NameNode on the same node, but due to the resource limitation, this was a scenario that we could not avoid.

On top of HDFS, we have YARN so that it is possible to execute data access or transformation tasks. Each of our nodes, that are already a DataNode, are also NodeManagers and, consequently, they have the ability to process tasks. Also on the YARN context, we require the ResourceManager to arbitrate all the resources in the cluster and effectively manage the distribution of tasks by the NodeManagers. Finally, on YARN, we have the role of the App Timeline Server that, very succinctly, serves the purpose of storing and presenting information about running and completed tasks.

Moving up on Hadoop's software stack, we have the components that implement Hive. Hive Metastore stores all the metadata concerning the database objects that were defined under Hive's scope, like tables and partitions. This information is stored in a relational database and in our case we opted to use a MySQL database, but other RDBMSs can be used. The WebHCat server provides an interface to Hive's metadata catalog management layer (HCatalog). HCatalog is built on top of Hive Metastore and provides a relational view of the managed objects that simplifies object management. Finally, HiveServer2 is the service that enables, preferably through JDBC, clients to execute their queries against Hive.

We have other services running in our cluster, some completely optional, like the Ambari Metrics that is responsible for collecting and processing metrics to help us manage the cluster, and others with more relevant functions like ZooKeeper that acts as a coordination service for distributed applications, or the Oozie Server, a workflow system that enable the scheduling of jobs in Hadoop.

After balancing the services throughout the cluster, the most relevant configuration task is to define several critical parameters that dictate how Hive, Tez, Map-Reduce and YARN perform their tasks. The list of parameters is extensive and requires, not only technical knowledge but also previous experience to determine their optimum configuration and should also take into account the data and processes that will be handled. To accomplish the parameter configuration, we used the script 'hdp-configuration-utils.py' that is provided by Hortonworks (2017). Due to our limited hardware, the default configuration gathered from this script proved not to be the best regarding performance. Through a set of tests, we concluded that the optimal configuration, for the allocation of YARN containers, should be between one and two containers for the same hard-drive. More than that hinders the performance of the tasks inside the containers, and so the script was changed to account for that limit. The final configurations set the minimum YARN container size to 2560 MB, and all the other parameters were generated, by the script, according to that value. With this configuration, we achieved the optimal configuration – Nodes 2 and 3 each can allocate two YARN containers and Node 1 has room for three containers, where the extra container, beyond the two, should be used for the allocation of the Application Master. In Figure 4.17 we present an overview of the cluster after the configuration of all its components.

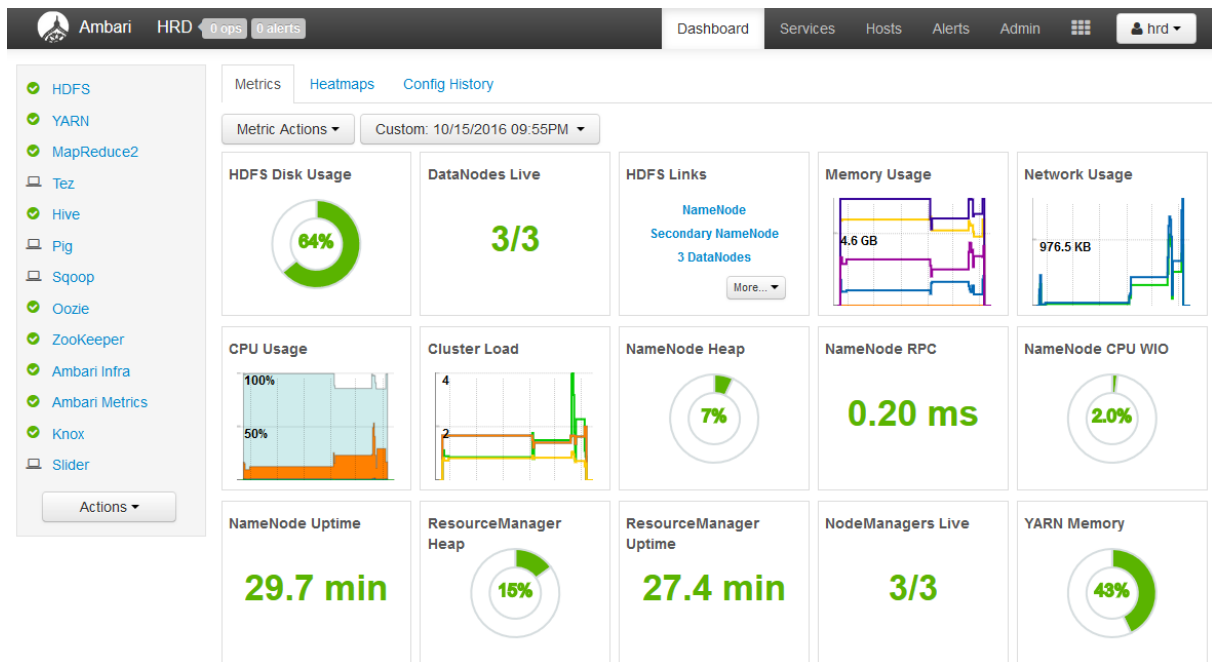


Figure 4.17. Ambari dashboard reporting the cluster overview

4.8.3. Data model design

The implementation of the data model in Hive replicated the same model implemented for the data warehouse, supported by the Oracle database, and presented in section 4.7.3. The minor differences between the two environments are related to the data types and to the partitioning techniques, but, in the end, they serve the same purpose.

4.8.3.1. Staging Area tables

The Staging Area tables in Hive map to the same exact files that were used to load data into the RDBMS. Like Oracle, Hive also can map tables directly over compressed files, a very useful feature that allows us to save large amounts of space in the file system. For the DDL related to the Staging Area tables, please refer to Appendix E.2.

4.8.3.2. Inventory tables

Like in the DW, the Inventory tables are partitioned by day and work as daily snapshots of the Mediaroom infrastructure. The contents of the Inventory tables are derived from the Staging Area tables, through a set of simple transformation processes, and stored in the HDFS using ORC file format that, according to our tests described in Appendix E.1, offer great compression ratios and no processing time overhead. The DDL and DML pertaining to the creation and loading of the Inventory tables is presented in Appendix E.4.

4.8.3.3. Fact tables

Both our fact tables are created as they were for the DW, but with a slight difference, the partitioning for the table `FACT_EVT_SEGMENTED`. In the RDBMS, this table was treated like a “temporary” table that only stores data as an intermediate step before the aggregations and afterward its contents are deleted. This approach was motivated by the enormous amounts of storage used by this table but in Hive, with the ORC file format, the storage requirements are greatly reduced, and so we can store this table permanently (we develop on this subject in section 5.4). To this effect, the table `FACT_EVT_SEGMENTED` is partitioned by day and by event type in Hive, while in Oracle it was partitioned just by day.

The DDL and DML related to the creation and loading of the fact tables is disclosed in Appendix E.5.

4.8.3.4. Aggregation tables

The summarized data, gathered from our fact tables, is stored in Hive as aggregation tables with the same structure as the ones created initially for the DW. We decided not to use the ORC file format here to facilitate the exchange of data between the cluster and the DW. One simple and viable approach to accomplish this is to simply map the text files generated by Hive as external tables for Oracle and the measurements calculated by the cluster would be immediately available in the DW. Additionally, not using the ORC file format for the aggregation tables does not affect the storage requirements since these tables have very few records.

The DDL and DML related to the aggregation tables is available in Appendix E.6.

4.8.4. Data transformation

After the creation of the underlying data model in Hive, we have developed the processes described in section 4.5.6 and already implemented in the RDBMS supporting the data warehouse (section 4.7.4). Our purpose was to replicate, in HiveQL, the statements already written in PL/SQL, rather than to write them completely from scratch. We have chosen this approach so that we could assess the effort associated to the porting of code from one system to another. We used HiveQL exclusively, without any user-defined functions, and we also did not attempt to use HPL/SQL because it was not available when we initiated our study, even though its assessment represents a very interesting opportunity that should be explored in future studies.

As the initial SQL statements were written using the ANSI notation of performing *joins*, rather than the specific Oracle notation, they could be completely re-used without any changes. The biggest difficulty we found was associated with the limited number of functions available to deal with date operations and, therefore, to cope with this, we had to rely on conversions to and from Unix time¹⁷. The generated statements were initially assessed regarding their execution plans, with the special concern to evaluate whether or not the partitioning scheme was being accounted for, and secondly to guarantee that the best Hive *join* techniques were being applied, like the *map joins* that are especially useful for the *star-*

¹⁷ Unix time describes time as the number of elapsed seconds since 1970-01-01 00:00:00.

schema joins where one big table is joined against several small tables. We have to note the enormous optimization improvements since earlier versions of Hive. Using the CBO and keeping the table statistics up to date, there was not a single time when we had to force more efficient execution plans through the use of SQL *hints*.

During the implementation and testing of the transformation processes, we also focused our attention on the DAGs associated to each statement and how parallelism was being applied by the creation of multiple Mapper and Reducer tasks. The DAGs responsible for the execution of each statement are presented in Appendix E.7.

Special care should be taken when using analytical functions, like *ranking*, since they will force Hive to process the data with a single Mapper task and, therefore, completely remove the usage of parallel processing. These functions should, whenever possible, only be applied to the already filtered results. If it is true that we can use the same SQL in Oracle or Hive, it is also true that, to optimize its execution, we need to be aware of the architectural differences between both systems and how the processing takes place.

The re-written SQL statements, associated with the data transformation processes being analyzed, are presented in Appendix E.5 and Appendix E.6.

4.9. SUMMARY

This chapter covered the design and development phases of our research where new technologies, from the spectrum of Big Data, are incorporated in a traditional data warehouse with the purpose of solving a specific problem – the calculation of television audience measurements from the raw data generated by the Mediaroom platform.

A database model and a set of transformation processes were designed from the initial in-depth analysis of the IPTV Mediaroom platform and implemented in the two environments specifically created to support our study – a DW supported by an Oracle RDBMS and a Hadoop cluster using primarily Hive on Tez. These are the environments that will be used, in the next chapter, as a testing ground for a subset of processes with the purpose of assessing the performance, scalability and storage capabilities that characterize each of the underlying technologies of the two systems.

5. EVALUATION

5.1. INTRODUCTION

This chapter is dedicated to the benchmarking of both the RDBMS and the Hadoop cluster while executing the same, previously implemented, transformation processes. This empirical assessment covers not only the performance of the systems, as they were initially defined, but also the evaluation concerning their scalability potential. Also, as performance is tightly connected to the size of data, we compare the storage requirements of each environment as they process and store the same data. The purpose of these benchmarking tests is to assess the performance of the new proposed architecture, using Hadoop, in comparison to the RDBMS.

We finalize this chapter with a section dedicated to the presentation of the calculated audience measurements and a brief assessment of the latest developments in Hive interactive query engine, the Low Latency Analytical Processing, and how it compares against an RDBMS when it comes to data access latency that is critical in supporting the visualization layer of data warehouses.

5.2. PERFORMANCE

The performance tests cover the transformation processes previously described and implemented on the RDBMS and the Hadoop cluster. The diversification of the processes, subject to our benchmarking, allow us to deepen the understanding of how distinct scenarios behave in both environments. Our generic goal is to assess if the Hadoop cluster can outperform the RDBMS in a set of transformation processes. However, beyond that we are trying to understand which processes fit better under the distributed architecture of a Hadoop cluster and from these conclusions collect valuable information that will support an efficient evolution of data warehouse architectures through the inclusion of Big Data technologies.

The execution of the benchmarking tests, presented in the next sections, was performed in completely idle systems where no other processes were running. Both our environments are running through virtualization, and to these specific tests their corresponding configuration is as it follows:

System	Software	Hardware (virtual)
DW	Oracle Database 12c Enterprise Edition	CPU: 2 cores Memory: 8 GB (Maximum memory target: 5 GB) Disk: WDC WD30EZRX (5400 RPM SATA-III)
		Node 1 CPU: 2 cores Memory: 10 GB (7.5 GB for YARN containers) Disk: WDC WD10EZEX (7200 RPM, SATA-III)
Hive	Hive on Tez using HDP 2.5.3	Node 2 CPU: 2 cores Memory: 6 GB (5 GB for YARN containers) Disk: WDC WD6400AAKS (7200 RPM, SATA-II)
		Node 3 CPU: 1 core Memory: 6 GB (5 GB for YARN containers) Disk: WDC WD30EZRX (5400 RPM SATA-III)

Table 5.1. Environment configuration for the performance tests

The benchmarking results presented in the next three sub-sections were obtained through the sequential execution of five iterations of the same scenario, where each scenario, for a given test, corresponds to the amount of data being used. From the five executions, the best and the worst results were removed, and an average was performed with the remaining three results. The results reported for each of the scenarios, within each test, are then a trimmed mean calculated by the following formula:

$$\mu = \frac{\sum_{i=1}^N (f(x_i)) - \text{Max}(\{f(x_1), \dots, f(x_n)\}) - \text{Min}(\{f(x_1), \dots, f(x_n)\})}{N - 2}$$

Where:

N is the number of iterations (5 in our case)

i is the iteration number

x is the test scenario

f(x) is the execution time of the test scenario

μ is the average execution time of the test scenario (trimmed mean)

Equation 5.1. Formula to calculate the average execution time of a test scenario

Each of the five transformation processes being benchmarked have a total of ten scenarios that correspond to the different number of data rows being fed to the transformation (i.e., one million, two million, up until ten million) and for each scenario, a total of five iterations are performed. The detailed execution statistics of the transformation processes are presented in Appendix F.

All the tables that are part of the transformation processes are analyzed before their utilization. This assures that, for both environments, the Cost Based Optimizer has the most accurate statistics and consequently is able to generate the most efficient execution plans.

5.2.1. Subscriber events transformation

The subscriber events transformation, as described in section 4.5.6.1, is characterized by a set of processes with the purpose of enhancing raw data entering the data warehouse. They represent the type of processes where the number of input rows is the same as the number of generated rows. Inside this group of processes, we cover three distinct transformations that vary in complexity and in the volume of the data being used.

The first transformation process, the *Channel Tune*, uses as input the raw files and performs two *joins* with two dimension tables, one small and another classified as medium. Also, due to its simplicity, this process does not use any Reducer task when executed in Hive.

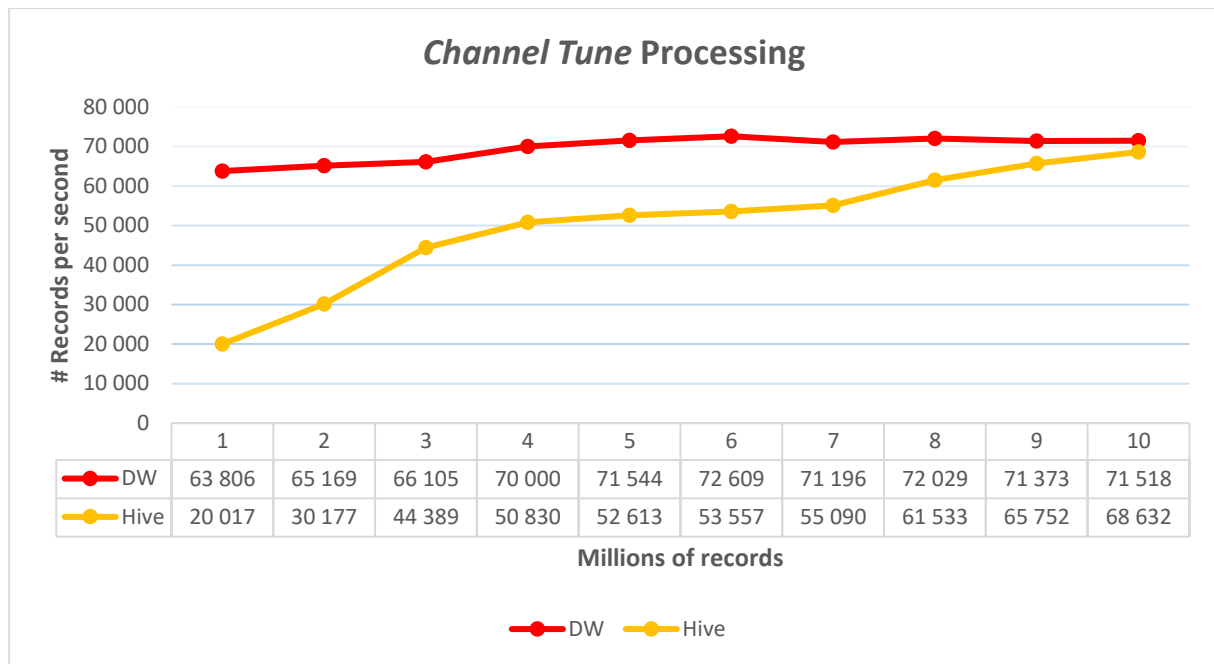


Figure 5.1. *Channel Tune* transformation benchmarking

From the analysis of the chart above, we can gather that the execution in the RDBMS (identified as DW) is more or less steady no matter the volume of data, while the same executions, in the Hadoop cluster (identified as Hive) are progressively increasing the number of records processed per second as the volumes of data increase. It is important to highlight that since we are using compressed files as input, the processing in the cluster cannot split them into multiple Mapper tasks, and for that, in the first two scenarios, only two nodes are being used to handle the raw data subject to the transformation. When we reach the scenario of ten million of rows, corresponding to ten distinct input files, we then have a total of ten Mapper tasks dedicated to the processing of the raw data.

As a final observation we can state that, for the amounts of data being used, the results obtained by the Hadoop cluster were far from being great. The trend shows us that above ten million rows the performance of the RDBMS would be surpassed, but only marginally.

The next transformation process, the *Program Watched*, uses similar amounts of data as input but also uses the data transformed by the *Channel Tune* process as a lookup table, so for this process we are considering not only a *join* with a small dimension table but also a *join* with a large fact table (with around ten million rows), or partition of a fact table to be more exact. Here we are reading and writing from the same table, but from different partitions.

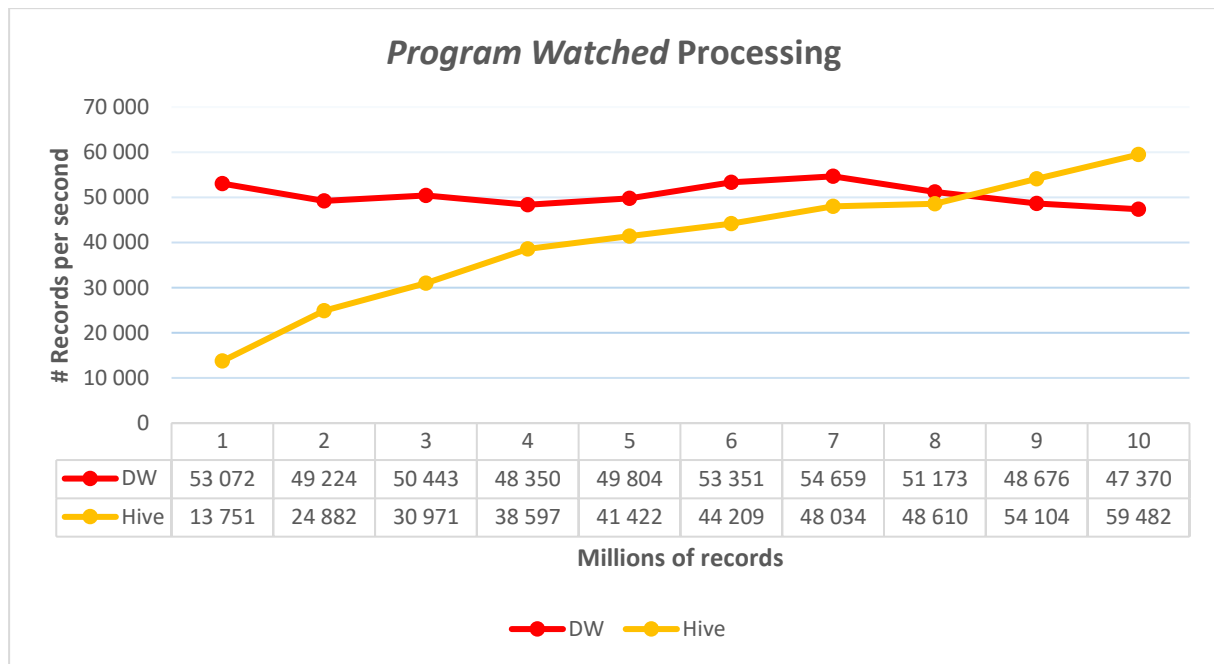


Figure 5.2. *Program Watched* transformation benchmarking

As expected and when we compare this process with the *Channel Tune* transformation, with the inclusion of a *join* with a large table, the transformation process gets its performance degraded. In Figure 5.2 we can observe that when we reach the nine million of rows in the input table, the cluster outperforms the RDBMS and the visible trend appears to show us that from this point forward, while the performance in the cluster continues to increase, the opposite happens in the DW. Again, we need to highlight that the cluster usage is dependent on the number of input files (since they are compressed) and, therefore, the number of Mapper tasks being created is directly related to the number of files. For the observed process, it is only possible to start using the three nodes when we reach the three million of rows that correspond to three distinct source files.

The following test uses similar amounts of data as input, but globally the amount of data is far less since we are not performing any *join* with a large table. The *DVR Events* processing, when compared to the *Program Watched* transformation, decreases the amount of data used as lookup and, when compared with the *Channel Tune* process, adds more complexity to the transformation statement by including more *joins*.

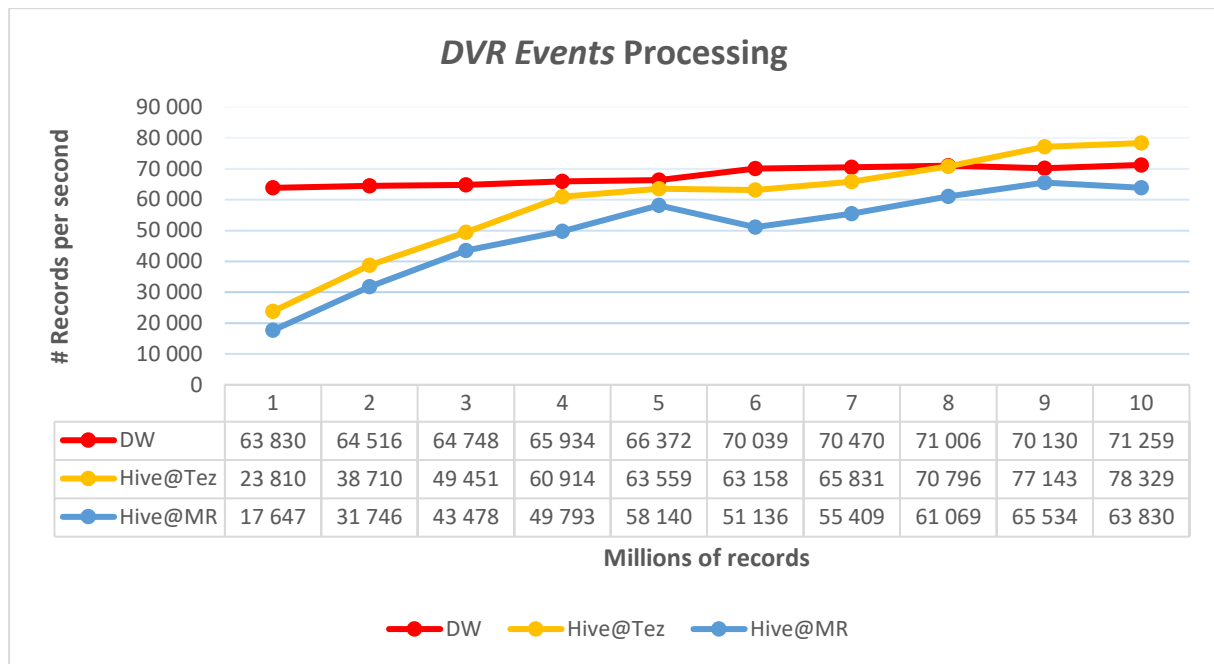


Figure 5.3. *DVR Events* transformation benchmarking

With just a quick glance at Figure 5.3, and especially if we compare it with the results in Figure 5.1, we can safely say that the extra complexity added to the transformation had no impact on the performance. For the DW, both transformations display similar performance values no matter the scenario. It is also true that, contrary to what happened for the *Channel Tune* transformation, here the Hadoop cluster was able to outperform the DW at the last two scenarios and the trend is still moving upwards. Also here, we took the opportunity to test the performance differences between Map-Reduce and Tez and the conclusion is that, as analyzed during the literature review performed at the Theoretical framework chapter, Tez offers consistently better performance than Map-Reduce, even on such a small test as the one conducted here. We will again take the opportunity to compare Tez and Map-Reduce on a test that intends to transform larger amounts of data.

These three transformation processes fit into the same category of processes that, in a simplistic view, take one input row, add information to it according to a set of rules, and finally output it to a fact table. The observations gathered for the RDBMS tell us that performance for these systems is more or less stable no matter the size of the input data, meaning that we can obtain excellent results with small amounts of data. On the other hand, the Hadoop cluster seems to thrive on the size of data. In our tests we did not reach a point where the cluster's performance suffered from degradation due to the amounts of data, by the contrary, the trend only showed us that the more data, the better.

As stated, the tested processes fit into the same category, but the differences between them allow us to have a deeper level of understanding regarding their behavior in different settings. Straightforward transformations, no matter the complexity of the lookup component, displayed solid results for the RDBMS, while a transformation that relies on large amounts of data to perform the lookup component (the *Program Watched* transformation), is the best candidate to collect the benefits of distributed processing, especially when the amounts of data increase.

5.2.2. Subscriber events segmentation

The *Event Segmentation*, described in section 4.5.6.2, is not a very typical process in the sense it performs a *cartesian product* that multiplies the number of input rows, according to a set of rules, and produces an output far larger than the input. For this reason, the costly step associated with the execution of this process is the writing of the output to the storage.

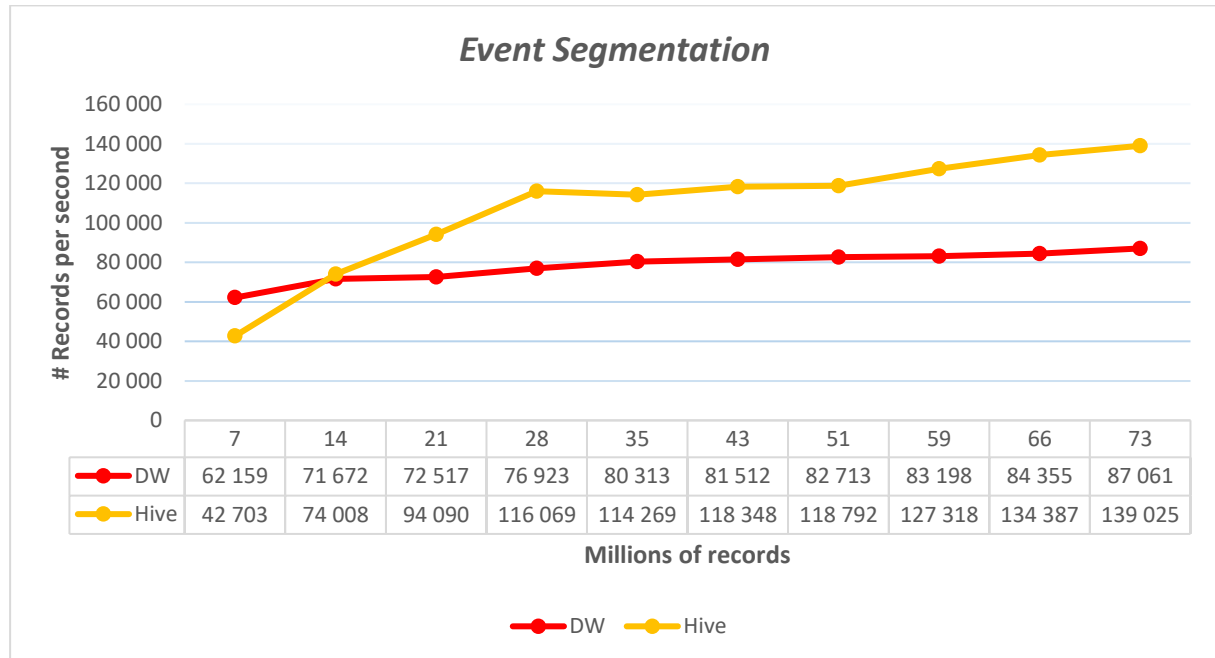


Figure 5.4. *Event Segmentation* benchmarking

The input data used by this transformation is the output generated by the *Channel Tune* process. Contrary to the processes tested previously, here the input data, in Hadoop, is not stored in compressed files but instead is composed of files using the ORC format. Due to this, the input readers can split the files by stripes and allocate the corresponding Mapper tasks. We did not force any number of tasks per Mapper but instead relied on Tez to automatically define the needed tasks according to the statistics calculated for the input files. These files that were previously generated by the *Channel Tune* transformation are small (less than 40 MB on average) and for that, and the current configuration of the cluster, one Mapper task is assigned to each file. Depicted in Figure 5.4, the test scenarios start with 7 million records (one single input file) and finish with 73 million, distributed by ten different files. Respectively, the allocation of Mapper tasks, for the processing of these records, ranges between one and ten. When we reach the third test scenario, and with the use of three Mapper tasks, the cluster is already greatly outperforming the DW, and by the forth, this difference is even larger. From this point forward the performance gains more or less stabilize, even though they start again gaining momentum in the last test scenarios. Throughout all the test scenarios, the RDBMS performance increases steadily, but slowly, never being able to get closer to the performance extracted from the cluster. This test revealed large performance gains by using the Hadoop cluster and also gave us the insight that, beyond four Mapper tasks, the performance gains tend to be less significant, albeit becoming relevant again with eight Mapper tasks or more.

Our Hadoop cluster is fairly limited in resources and is also composed of heterogeneous nodes where concurrency of YARN containers can impose significant overhead. Another aspect the can condition

the performance in our cluster is in which node the Application Master is created. The resource constraints of the cluster supporting this study, and in particular for this more complex test, was a useful mean towards gaining insights on how the correct resource sharing and concurrency can be used to optimize performance.

5.2.3. Subscriber events aggregation

The subject of our final test is not a process that is part of the ETL layer of a data warehouse but instead, it belongs to the more analytical components. The concept of transformation still applies here but more in a literal sense than in a conceptual one. Data is indeed transformed, but that transformation is performed through analytical capabilities on top of an aggregation. Since our data warehouse architecture uses ELT rather than ETL, transformation and aggregation processes co-exist in the same system. This gives us the opportunity to explore the analytical performance capabilities of a distributed architecture easily. The *Audiences Aggregation* process makes use of the data previously generated by the *Event Segmentation* transformation, and its purpose is to calculate aggregated measurements related to television audiences.

Unlike the other processes analyzed so far, here the depicted number of records per second in Figure 5.5 reflects the number of input records instead of the number of output records. This process represents the case where the number of input rows is far greater than the number of output rows, as it is characteristic of aggregations.

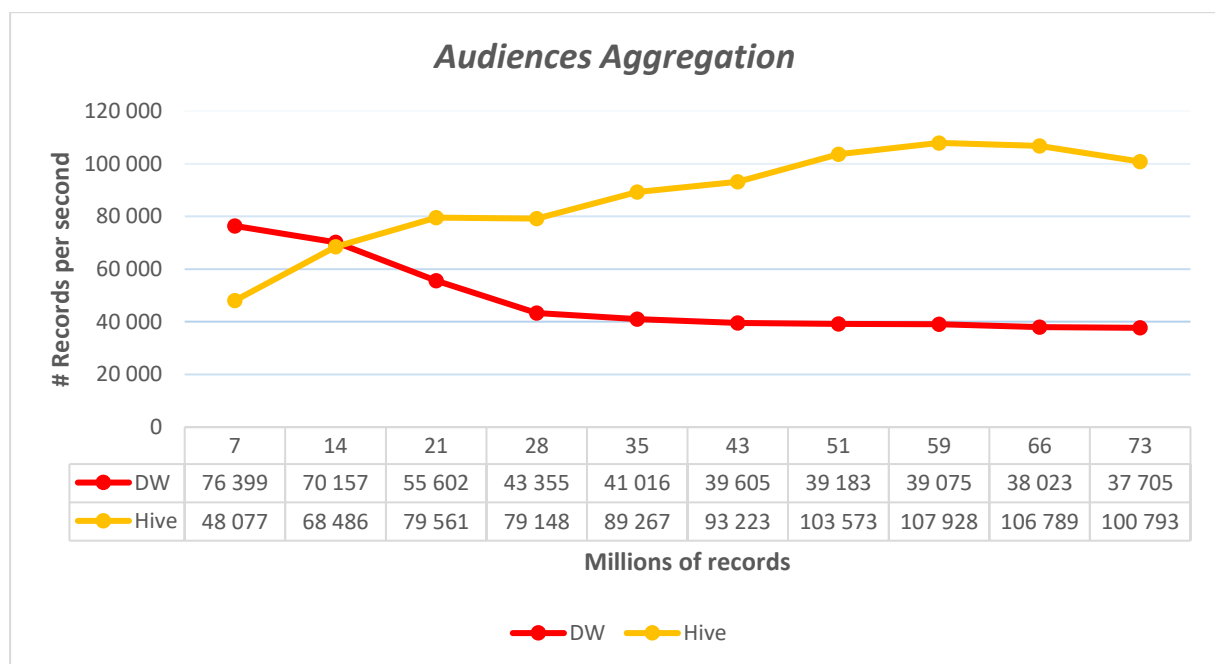


Figure 5.5. *Audiences Aggregation* benchmarking

From the analyzed processes, this is the first where the amount of data seriously affects the performance of the RDBMS. In Figure 5.5 we can observe two opposite trends – the DW performance degrades, with the increase of input data, while the performance of Hive improves.

For this process, we also performed several tests using Map-Reduce instead of Tez but the results obtained were very disappointing. The execution times were completely diverse, ranging from 1300 seconds to more than 7000 and for multiple occasions, the jobs simply failed to finish. Also important to highlight is that the best time obtained with Map-Reduce was still two times slower than the worst execution time on Tez.

As a retrospective of the performed tests, and their corresponding benchmarks, this process is where we were able to obtain the greatest performance gains by the use of Hive on Tez, reaching almost results that are three times better.

5.3. SCALABILITY

Data warehouses fulfill their mission by collecting data from multiple sources, transforming it into meaningful information and delivering actionable knowledge that can be used to improve business processes and thus adding value to the organizations. Data plays a key role in this chain of value and the ability to process it relies on the capabilities of the DWs. As data increases in volume and in the velocity in which it is generated, capabilities and their underlying resources need to be expanded so that systems remain viable in their mission. The expansion of capabilities can be approached from several perspectives, surrounding different aspects, of both the problem and the solution. In this section of our research, we were interested in understanding, empirically, the performance gains of scaling-up the RDBMS versus the scaling-out of our Hadoop cluster. Scaling the resources is a fast and simple route to evolve a system so that it continues to be able to fulfill its mission.

In this section, we scale both systems that were used during the performance tests (section 5.2) and benchmark once again the execution of our tests. The hardware configurations for the systems, tested in this section, are described in Table 5.2.

System	Software	Hardware (virtual)
DW	Oracle Database 12c Enterprise Edition	CPU: 2 cores Memory: 8 GB (Maximum memory target: 5 GB) Disk: WDC WD30EZR (5400 RPM SATA-III)
		Node 1 CPU: 2 cores Memory: 10 GB (7.5 GB for YARN containers) Disk: WDC WD10EZEX (7200 RPM, SATA-III)
		Node 2 CPU: 2 cores Memory: 6 GB (5 GB for YARN containers) Disk: WDC WD6400AAKS (7200 RPM, SATA-II)
Hive-3N	Hive on Tez using HDP 2.5.3	Node 3 CPU: 1 core Memory: 6 GB (5 GB for YARN containers) Disk: WDC WD30EZR (5400 RPM SATA-III)
		Node 4 CPU: 1 core Memory: 6 GB (5 GB for YARN containers) Disk: WDC WD30EZR (5400 RPM SATA-III)
DW-X	Oracle Database 12c Enterprise Edition (with Parallel Auto)	CPU: 6 cores Memory: 24 GB (Maximum memory target: 20 GB) Disk 1: WDC WD10EZEX (7200 RPM, SATA-III) Disk 2: WDC WD30EZR (5400 RPM SATA-III)
Hive-4N	Hive on Tez using HDP 2.5.3	Node 1 CPU: 2 cores Memory: 10 GB (7.5 GB for YARN containers) Disk: WDC WD10EZEX (7200 RPM, SATA-III)

	CPU: 2 cores
Node 2	Memory: 6 GB (5 GB for YARN containers)
	Disk: WDC WD6400AAKS (7200 RPM, SATA-II)
	CPU: 1 core
Node 3	Memory: 6 GB (5 GB for YARN containers)
	Disk: WDC WD30EZR (5400 RPM SATA-III)
	CPU: 1 core
Node 4	Memory: 6 GB (5 GB for YARN containers)
	Disk: SanDisk Ultra (USB3)

Table 5.2. Environment configuration for the scalability tests

The DW was expanded to 6 CPU cores, a total of 24 GB of memory and a second hard drive to reduce concurrency during input/output tasks. To better harvest the benefits of the hardware upgrade, the system was adequately configured by re-defining the memory parameters and enabling parallel processing to a maximum of four threads per process. The new system that resulted from this upgrade is referred as “DW-X” in the next tests.

Our Hadoop cluster was scaled-out by adding a new node with data processing ability (DataNode and NodeMaster). We also changed the replication factor from three to four and updated this attribute for all the existing files. This assured us that any of the nodes had the ability to process any chunk of data. The “new” cluster was upgraded to a total of 28 GB of memory and 6 CPU cores, and it is referred as “Hive-4N”. The cluster used during the performance tests is referred, in this section, as “Hive-3N”.

For our scalability tests, we opted to consider a subset of the processes previously benchmarked. This subset works in sequence, being the output of one process the input of the next, and covers the three major transformation process types being studied, regarding the relation between the number of input rows and the number of output rows (*One-to-One*, *One-to-Many* and *Many-to-One*). For the detailed execution statistics, please refer to Appendix G.

The first evaluated process is the *Channel Tune* and it depicts a *One-to-One* transformation type.

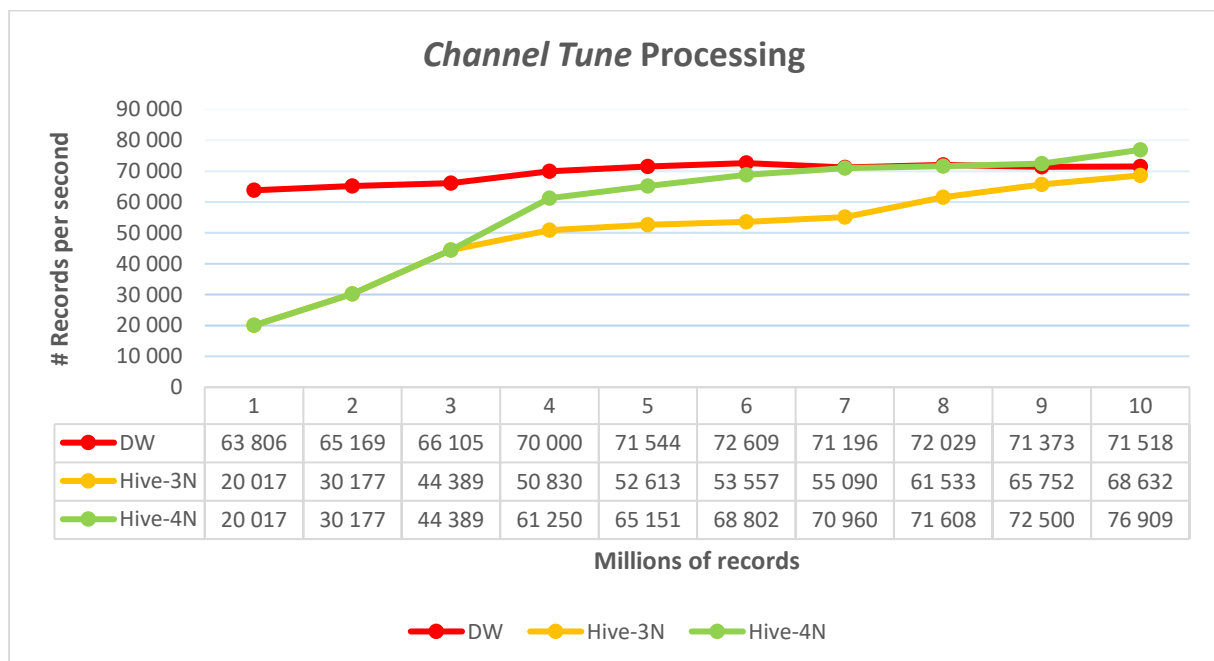


Figure 5.6. *Channel Tune* transformation scalability benchmarking

The *Channel Tune* transformation was the only test executed previously where the cluster was not able to outperform the RDBMS. This was due to the simplicity of the process where the results are produced between fifteen seconds and a little over two minutes, but nevertheless, we used the scaled-out cluster to assess if we could surpass the RDBMS performance.

Due to the already mentioned fact that parallelism for this test is directly related to the number of input files, the measurements for the scaled-out cluster only start at the fourth test scenario and, therefore, the results displayed in Figure 5.6 include, for the first three test scenarios, the previously collected measurements for the cluster with three nodes as being also the results for the cluster with four nodes. From the fourth test scenario onwards, we can observe that the expanded cluster shows significant performance gains when compared to the previous cluster. By the end of the test scenarios, our new cluster can outperform the RDBMS. It is true that these improvements are not very substantial, but this test ultimately serves the purpose of demonstrating that scaling-out is a simple solution to cope with larger amounts of data.

The *Event Segmentation* transformation test makes use of much more data than the *Channel Tune* process, and previously we already were able to collect significant performance gains with the Hadoop cluster. For this test, we compare not only the scaled-out cluster but also the scaled-up RDBMS.

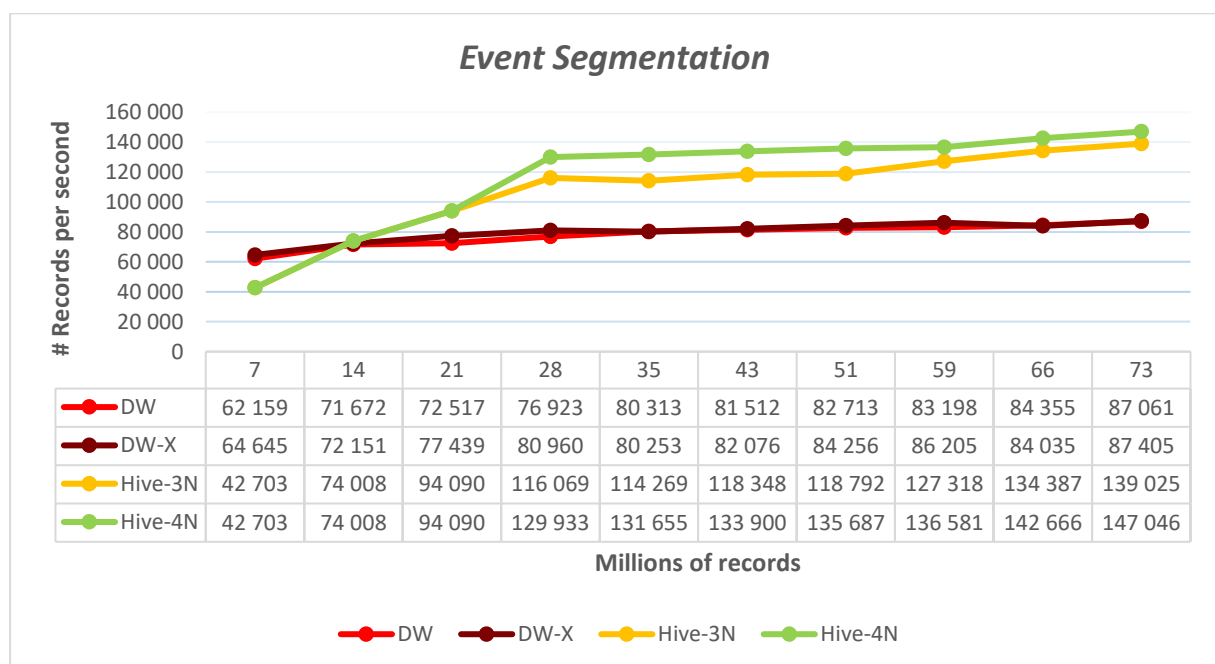


Figure 5.7. *Event Segmentation* scalability benchmarking

By analyzing the chart presented in Figure 5.7, the first observation we gather is that the scaled-up DW added no performance improvements. The reason behind this behavior lies within the process itself. The *Event Segmentation* transformation is a process that relies heavily on the speed of writing to the disk. The reading of information and the complexity of the transformation add very little cost to the transformation when we compare it with the cost of writing the large amounts of data to the disk. This is why the scaled-out cluster consistently gives us performance improvements. In this cluster, we have four disks instead of three, even though we have to deal with the overhead of having the replication factor set to four.

Our last scalability test takes to the heaviest of the processes under scrutiny, the *Audiences Aggregation*. Previously this was the transformation process where the cluster greatly surpassed the RDBMS, and here we benchmarked the upgraded systems to assess the impacts of scalability in both architectures for a process that relies more heavily on disk reads and memory.

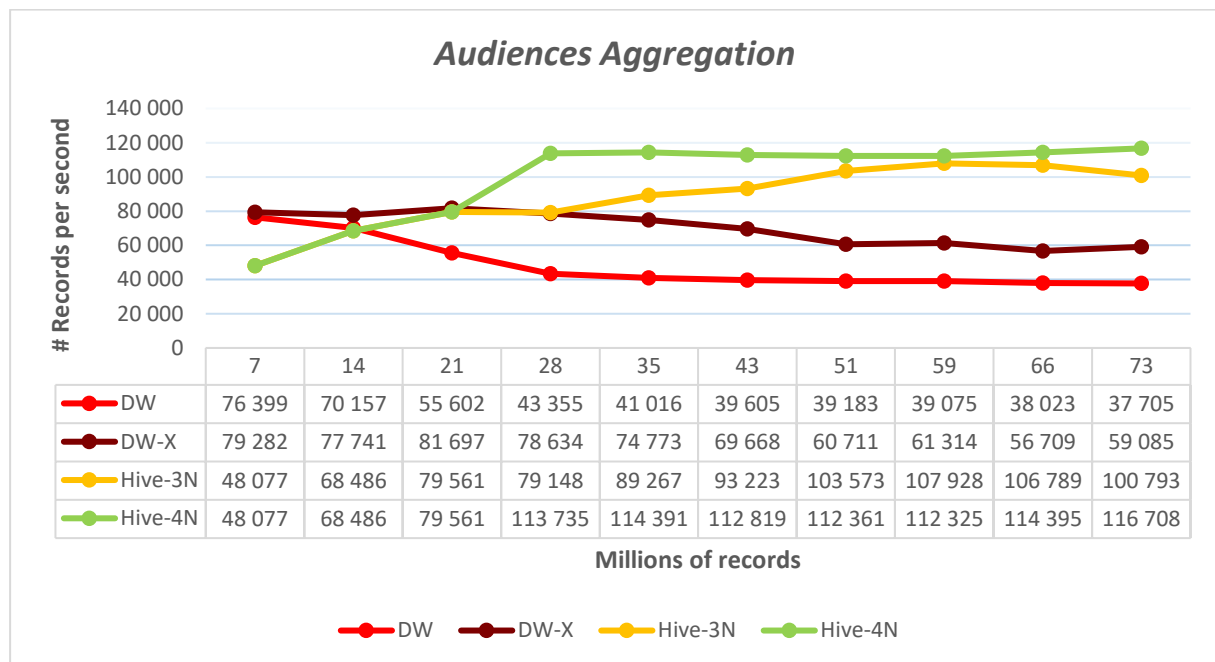


Figure 5.8. *Audiences Aggregation* scalability benchmarking

The results of both the expanded systems, displayed in Figure 5.8, outperformed their original systems, but it is very evident that the distributed processing is clearly the best fit for heavy aggregations, like the one subject to this test. For the RDBMS to come close to the performance of the Hadoop cluster, the level of scaling-up would have to be enormous.

From the analysis of the scalability test results, one thing became clear – the scaling-out of the cluster, no matter the type of process, always give us visible performance gains, while tangible improvements of scaling-up the RDBMS are dependent on the process. This final observation refers us to a potentially serious problem surrounding the capability of data warehouses to cope with the increase in the volumes of data. There may be processes that would require more costly and complex hardware improvements to retain their validity when facing larger volumes of data.

5.4. STORAGE

During our storage analysis options, performed in Appendix D.1 and Appendix E.1, we already observed that Hive supports file formats capable of storing data with high compression rates, without hindering performance, especially through the ORC format.

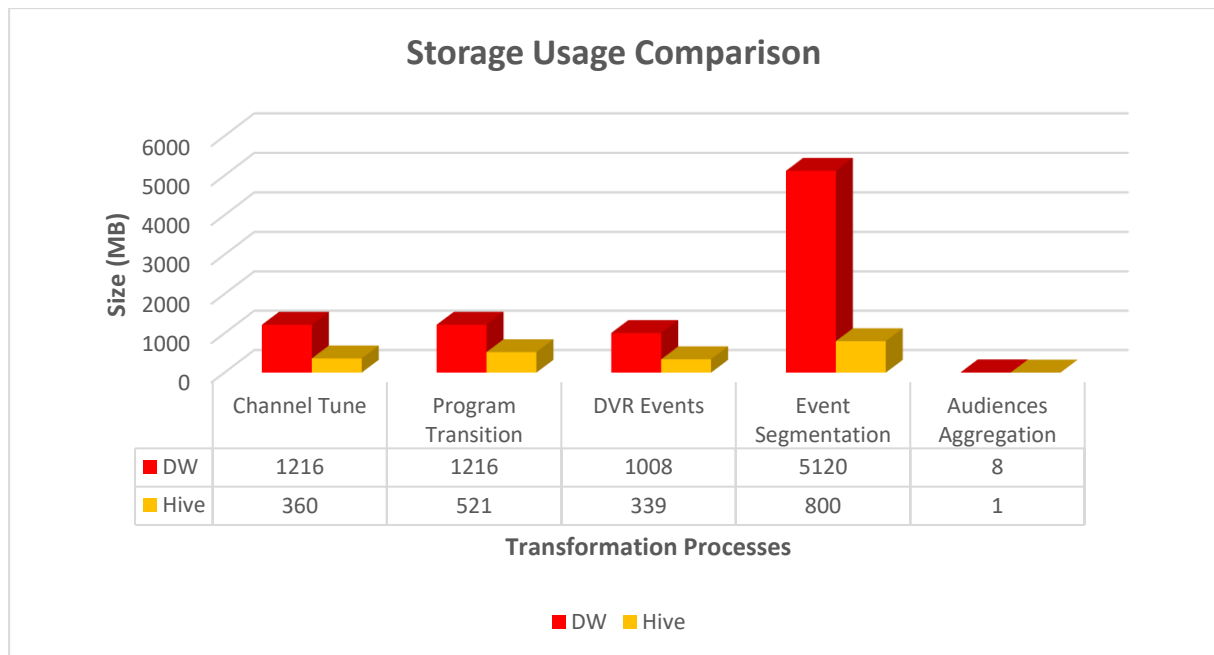


Figure 5.9. Storage usage comparison

In the chart above we present the comparison of storage requirements in both the RDBMS (with *basic compression*) and the Hadoop cluster (with Hive’s ORC format without any extra compression). We have compared the storage usage for each of the tested transformation processes with the maximum amount of data used during the tests and the results tell us very clearly that, even without any compression algorithm, the ORC format enables huge savings in the storage space utilization.

5.5. DATA VALIDATION

With the purpose of assuring that the data being transformed on both systems is exactly the same, we performed a set of tests that targeted the outputs of each process. The data generated in the Hadoop cluster was exported to the DW and subsequently compared against the corresponding data that had been generated inside this system. This comparison exercise is fundamental not only to assure data quality but also to ensure that the recorded execution times were indeed considering similar processes in the sense that, with the same input, they generated the same exact output. Even though the transformation processes in both environments are very similar, they aren’t exactly the same, namely because of the differences regarding the data types and in the use of functions to manipulate dates.

For the validation tests, our decision towards the push of data from the cluster to the data warehouse relates to two important considerations. Firstly, comparison of tables within our RDBMS is easier because Oracle supports the SQL operator *minus* while in Hive the same operation can only be performed through *outer joins*. The second, and most important consideration, is founded around one critical aspect driving this study, the integration between the two systems. Our purpose is to optimize heavy transformation processes by migrating them from the DW to a cluster and subsequently move the calculated results back to the DW where they can continue to serve the reporting layer. To implement the data exchange between the two systems, we made use of Apache Sqoop, a tool designed for the bulk transfer of data between Hadoop and structured datastores. The use of Sqoop is rather simple, and its programming can be done by expressing the data transfers through SQL

statements. It supports direct export from several formats, like ORC, and includes a metastore where jobs can be stored for recurrent executions, a particular useful feature when dealing with multiple transformation tasks that are executed recurrently.

Through these data validation tests, we were also able to briefly explore a viable solution capable of assuring data exchange between the Hadoop cluster and the data warehouse.

5.6. VISUALIZATION

As part of data warehouses' global architectures, the *Data Access and Analysis* layer brings information closer to the users through several tools and techniques. One of the goals of our study is to deliver meaningful television audiences metrics that were extracted from the raw data generated by the Mediaroom platform. For that purpose, we have developed a set of reports that present some relevant television measurements and deliver quick insights to their consumers.

The reports were developed using Microsoft Reporting Services 2016 and are supported by the aggregation tables that were previously calculated by the Hadoop cluster and then propagated to the data warehouse's Oracle database. Since we have the same data residing in both environments, we also used this opportunity to perform some brief analyses regarding the latency of both our systems with the purpose of assessing whether or not Hive can be a feasible solution to support reports where the speed associated with the presentation of results is of the uttermost importance.

5.6.1. Reporting

The reports developed did not intend to offer a full and comprehensive view of the measurements that we can produce from the Mediaroom subscriber events. For demonstration purposes, we focused our attention around some of the best-known metrics in the field of television audiences, the *Rating*, the *Reach* and the *Share* of Live television. Here we present a dashboard that aggregates several measurements associated with a television channel. Its purpose is to deliver quick insights at the distance of a simple click.

The dashboard presented in Figure 5.10 provides a general overview of any given television channel for any day. This dashboard encompasses several aspects relevant for a TV channel and communicates clearly some of its pertinent metrics. On a single view, we have the daily *Reach* and *Share* of the channel, the detailed evolution of the *Share* throughout the entire day, the top ten programs watched with their corresponding *Rating* and rank, as well as the top five programs that were recorded and/or visualized as part of the Digital Video Recording service.

complemented by *inner joins* to the dimension tables to obtain descriptions like the channel or program names. It is also important to note that both the source tables and the outputted results consist of very small datasets that in most cases represent less than 1000 records.

Under this scenario, all the report queries executed against the RDBMS took less than one second to complete, and, therefore, the reports are presented immediately. On the other hand, the same executions, but this time against Hive (using Tez), took between thirty seconds and one minute and a half to complete. Since Tez can cache intermediate data and even results inside the same session, we can reduce the execution time of the queries, to just some seconds (between 2 and 15), if the queries are re-executed inside the same session. However, in our current scenario, where the reporting tool does not re-use the same session, we could not collect any benefits from Tez caching mechanisms.

From the results of these tests, it is obvious that latency on Hive is too high and, therefore, we cannot display the reports in an interactive way. We expanded our tests further and decided to test Hive's new feature regarding the interactive querying subject, the Low Latency Analytical Processing (LLAP) component. LLAP uses a combination of in-memory caching, pre-fetching and persistent query executors (similar to Impala's long-lived daemons) and enables a hybrid execution model, where large queries continue to be processed in standard YARN containers, while the persistent executors handle small queries. Using LLAP, the execution of our reports for the first iteration was cut in half and for the subsequent executions always performed under ten seconds with results similar to the ones already obtained by the re-execution of queries inside the same session just by using Tez.

Using LLAP, we can query small datasets much faster but currently still not even close to the performance of an RDBMS. Moreover, enabling the LLAP adds overhead to the cluster that, depending on its size, can severely hinder its overall performance. On our tests, LLAP permanently allocated three YARN containers and by doing so, our total of seven possible YARN containers was greatly reduced and thus also the possibility of parallel processing when performing transformation tasks.

5.7. SUMMARY

From the previously implemented systems, this chapter was dedicated to the exploration of both technologies with the purpose of empirically discerning their strongest and weakest aspects. On a general perspective, our tests focused on three main aspects – data manipulation, data storage, and data access. On top of these characteristic functions of database management systems, we have assessed the scalability potential of each system and their associated performance benefits. This is a critical aspect in assuring the viability of any system, especially in scenarios where the volume of data tends to increase significantly.

Finally, our evaluation took us to the data warehouse visualization layer with the purpose of delivering the calculated audience measurements and assessing the interactive querying performance of both systems, a fundamental aspect for the delivery of reports to the users.

Triggered by the initial problem and supported by our theoretical framework, the benchmarking tests conducted in this chapter provided us a set of results and valuable inferred insights that will be subject to discussion in the next chapter as we move to integrate our findings in the definition of an enhanced data warehouse architecture supported by Hadoop.

6. RESULTS AND DISCUSSION

6.1. INTRODUCTION

This chapter is devoted to the presentation and discussion of the results gathered in the course of our evaluation phase. During the execution of our tests, we presented and commented the results associated with each specific observation, but here we present a more summarized view of these results and, associated to them, a more generalized discussion with the purpose of making the bridge from the specific problem to the general area of study, the data warehouse architectures.

6.2. PERFORMANCE

6.2.1. Batch processing

Hadoop was designed for parallel batch processing of large amounts of data (Barnes et al., 2016). On top of Hadoop, Hive offers a familiar SQL-like approach of implementing distributed data transformation tasks that fit the typical batch processing use cases (Grover et al., 2014). Also, at the storage level, there is a clear orientation towards batch processing since HDFS focuses on the overall throughput rather than the latency of individual operations (Shvachko, Kuang, Radia, & Chansler, 2010). Through our performance tests, we could effectively assess that the combination of these characteristics is, in fact, materialized in great performance gains during the execution of data transformation tasks. In Figure 6.1 we present the cumulative execution time of the benchmarked processes during our research (the hardware details of the systems are in Table 5.1). From the first look at this chart, the most obvious conclusion is that the same data transformation tasks in Hive are completed in around half the time that takes them to be processed in the RDBMS.

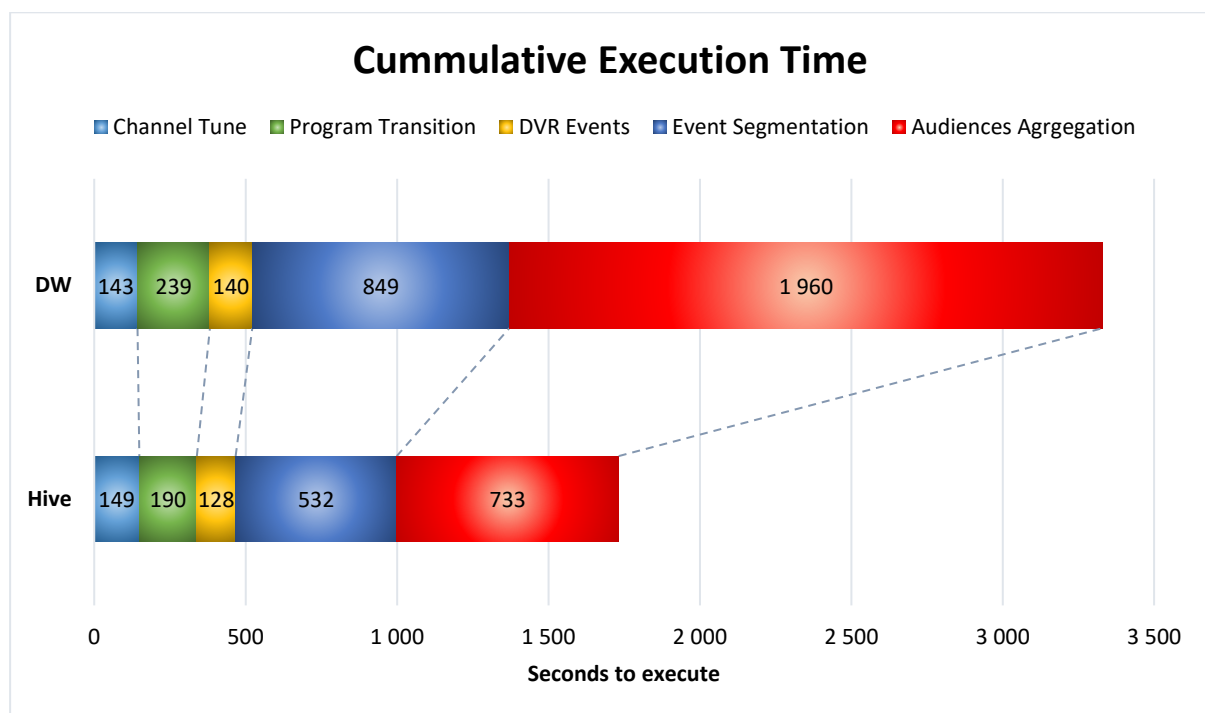


Figure 6.1. Data transformation tasks cummulative execution time

Figure 6.1 also gives us other quick insights – our classification of transformation processes, as depicted in section 4.7.4 (1:1, 1:M and M:1), confirmed us that not all data transformations are the same. The processes that fit in the *One-to-One* category show little or no improvement when processed by Hadoop, but, as the volumes of data grow, their performance increases to the point where eventually it surpasses the RDBMS (this is shown in section 5.2.1). The *One-to-Many* and the *Many-to-One* processes are where Hadoop outperforms, without any doubt, the RDBMS. One aspect that is common to all the processes is that the more data we have to process, the bigger is the performance difference between Hadoop and the RDBMS.

6.2.2. Interactive querying

We have tested the data access latency in Hadoop during the execution of the reports designed to show the calculated television audience measurements. The results are much in line with what it was to be expected according to the underlying design of the supporting technologies. As we discussed in the previous section, HDFS and Hadoop are designed for throughput rather than latency. All of the SQL statements feeding the data to the reports complete in under a second when we use the RDBMS while these same statements, using Hive, can take more than one minute to return the data. The subject of latency in Hadoop is being addressed through some initiatives, namely the Low Latency Analytical Processing in Hive, and indeed it can be reduced very significantly – using LLAP we could cut in half the execution of some statements and even, through caching, consistently deliver the data under ten seconds. If it is true that LLAP gives us faster access to data, it is also true that this comes with the cost; to provide faster results, LLAP uses persistent query executors, and this requires the permanent allocation of YARN containers that consume and remove resources from the batch processing capability.

Hive, even with LLAP, cannot match the low latency delivered by an RDBMS and consequently, data cannot be instantly displayed in the visualization layer, a very important feature, especially in the presentation of dashboards that use multiple datasets. Nevertheless, in recent years, the need for low latency solutions in Big Data systems has gained focus, namely through the emphasis in Lambda Architectures that discern the different purposes of data by layers – the *Batch Layer*, the *Serving Layer* and the *Speed Layer* (Marz & Warren, 2015). Today's fact is that Hadoop cannot match an RDBMS regarding low latency and thus it cannot effectively support interactive querying, but with Lambda Architectures gaining momentum and adoption we can expect for the minimization of this performance gap in a near future.

6.3. SCALABILITY

The first consideration about scalability is its simplicity when it comes to Hadoop; increasing or decreasing the processing power or storage capabilities of a cluster entails only the addition or removal of nodes. No further configurations are required. Scaling an RDBMS is far more complex, starting with the need to change the hardware configuration (i.e., adding more or better CPUs, disks or memory) and then reflecting these changes in the configurations of the database instance. In a virtual

environment, the hardware assignment does not have this complexity, but nevertheless, the database and even the operating system need to be re-configured if we want to maximize the upgrade benefits.

In our research, we have scaled both our environments to the maximum capabilities of the available hardware, and in the end, the processing power of both systems was very similar (the hardware details of the systems are in Table 5.2). In Figure 6.2 we present the results of our scalability tests where we compare the processing of three transformation processes in both the systems before and after the upgrade. The RDBMS is represented by the acronym *DW* and its scaled-up version by *DW-X*. The Hadoop clusters are referred by *Hive-3N* and *Hive-4N* where the numeric part indicates the number of nodes.

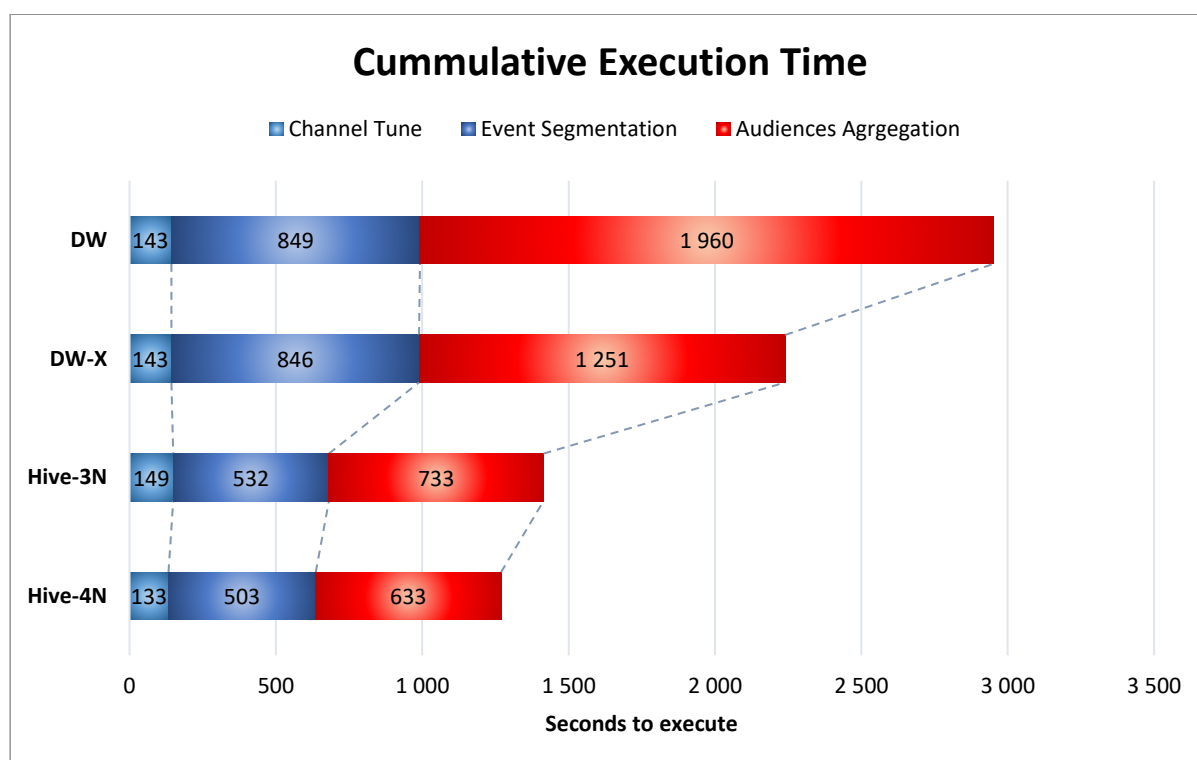


Figure 6.2. Scalability effects on the data transformation tasks execution time

From the information presented in the chart above we can infer several conclusions. Firstly, we got improvements from scaling both systems. The improvement in the RDBMS is far greater (25% against 10%), but we need to consider that the scaling-up of the RDBMS tripled its resources while the scaling-out of the Hadoop cluster just added a new node to the three already in place. Even this slight upgrade resulted in improvements for all the processes, whereas in the RDBMS we only got improvements for the *Audiences Aggregation*. This process falls in the category of the *Many-to-One* and it is the one that requires the most memory. To perform *hash joins*, Oracle uses the PGA and what does not fit in memory is overflowed to the *temporary tablespace*. The memory upgrade, given to the RDBMS, greatly sped up this process because the overflow from memory to disk was greatly reduced. One feature of Oracle's architecture is that each session can only use a portion of the PGA and thus we cannot use it entirely to perform large *joins* (Oracle Corporation, 2017). In Hadoop, there are no limitations to dedicate all the resources of the cluster to complete a single process.

Besides being much easier to implement in Hadoop, our results showed that the scalability in a Hadoop cluster is far more efficient in making use of the resources. The execution time decreased 25% in the

RDBMS but, to achieve those results, we had to triple the associated memory (from 8 GB to 24 GB) and CPU cores (from 2 to 6), while the 10% execution time decrease in the cluster was obtained by just adding a new node with 6 GB of memory, to the existent 22 GB, and a single CPU core to the existing five allocated CPU cores. In the RDBMS, to get an improvement of 25% we had to increase the hardware by 200%, while the 10% performance increment in the cluster was obtained by just upgrading the hardware by less than 25%¹⁸.

Finally, scalability under the distributed architecture of Hadoop is not only easier to implement and more efficient in the resource utilization, but it is also virtually unlimited as it grows horizontally, through the inclusion of more nodes, instead of growing vertically under the hardware constraints of a single server.

6.4. STORAGE

From the analysis of the storage usage in both systems for the same data, presented in Figure 6.3, it is very obvious the enormous gains that are obtained by using Hive's ORC format, in comparison to the compression used in Oracle.

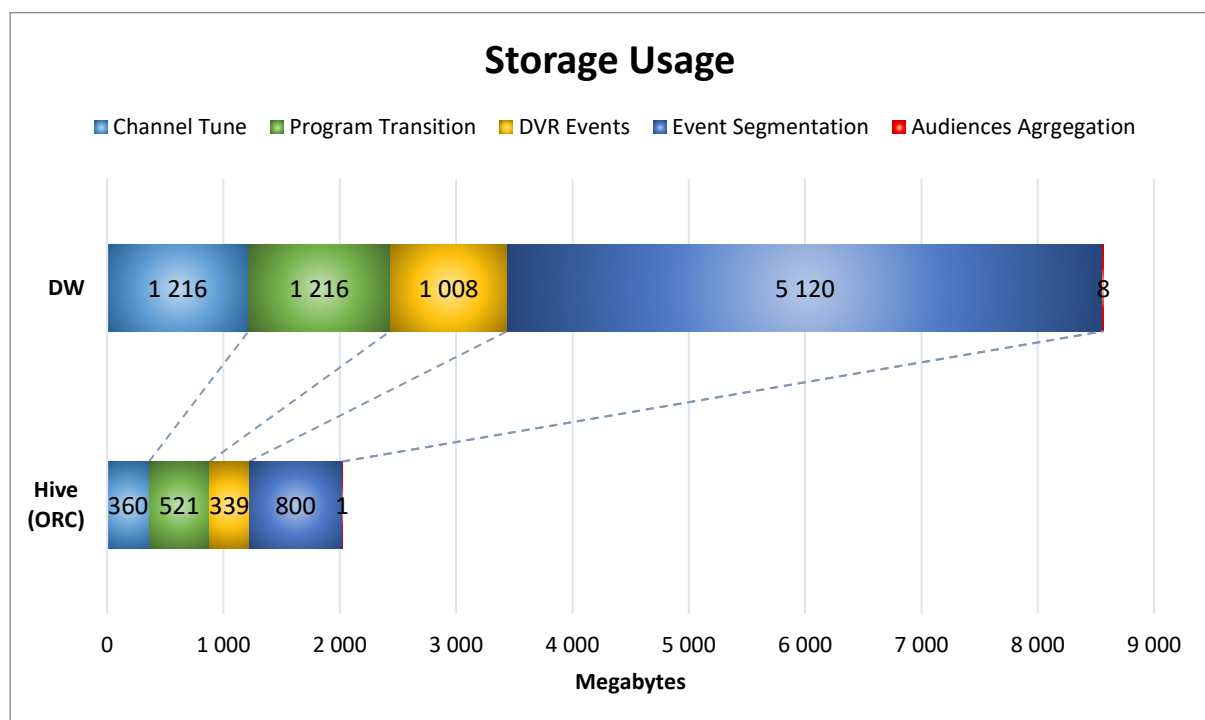


Figure 6.3. Total storage usage comparison

From the original size of data in Oracle, Hive's format allowed us to save 76% of storage space¹⁹. It is important to state though that we are comparing two very different storage approaches – Oracle *basic compression* is a row compression method while Hive's ORC file format uses columnar compression.

¹⁸ These comparisons only took into consideration memory and CPU, even though an extra disk was added to both environments.

¹⁹ We did not account for replication in the cluster nor for redundancy in the RDBMS.

We were forced to use row compression in Oracle because columnar compression is not available for the used database, such compression is only available in more high-end products like Exadata.

In Figure 6.3 we are showing the storage usage for Hive's ORC format without any extra compression, but if, for example, we added the Zlib compression, the default in Hortonworks Data Platform, the storage savings would improve from 76% to 91%. It is clear that Hive's ORC format is extremely efficient in storing data and associated to this efficiency is the performance of any read/write operations.

6.5. ARCHITECTURES

Data warehouse architectures are composed of a multitude of systems, and in turn, these systems have their own intrinsic architectures. In our research, during both the theoretical and empirical phases, we have explored several aspects of the underlying technologies and architectures of Relational Database Management Systems and Hadoop. From the knowledge gathered, we present in Table 6.1, the major advantages and disadvantages concerning Hadoop and RDBMSs architectures.

Technology	Pros	Cons
RDBMS	<ul style="list-style-type: none"> • Very mature and robust; • Has many features; • Deep support for relational semantics; • Deep support for transaction processing; • Extensive knowledge base; • Great for interactive querying; • A large number of skilled professionals. 	<ul style="list-style-type: none"> • Mostly proprietary software; • Licensing costs; • Data must be structured; • Little support for complex data structures; • Fault tolerance requires complex and expensive approaches (redundancy); • Optimized for high-end custom hardware; • Limited and expensive scalability.
Hadoop	<ul style="list-style-type: none"> • Open source; • None or little licensing costs; • Uses cheap commodity hardware; • Supports heterogeneous hardware; • Data structure is not mandatory; • Deep support for complex data structures; • Designed for batch processing; • Great for massive full scans; • Very fault tolerant. Fault tolerance is a critical part of its design; • Virtually unlimited scalability; • Advanced data compression. 	<ul style="list-style-type: none"> • Relatively new technology; • Complex architecture constructed by many components; • Limited set of features still under development; • Poor performance for interactive querying; • Limited knowledge base; • Limited support for relational semantics (Hive, Impala, etc.); • Little support for transaction processing; • A limited number of skilled professionals.

Table 6.1. Comparison between RDBMS and Hadoop architectures
(Adapted from Goss & Veeramuthu, 2013; Kimball & Ross, 2013)

The overview provided by Table 6.1 can be an extremely helpful guide towards aligning specific requirements with the most viable solutions. In our specific case, we were interested in enhancing the transformation layer of a data warehouse so that it could cope with large volumes of data while, at the same time, assuring a responsive visualization layer that could quickly deliver the results to the users. From our study, and summarized in Table 6.1, we verify that currently, the best approach is to integrate both technologies in the data warehouse architecture, and thus we present in Figure 6.4 the high-level diagram of the proposed new DW architecture.

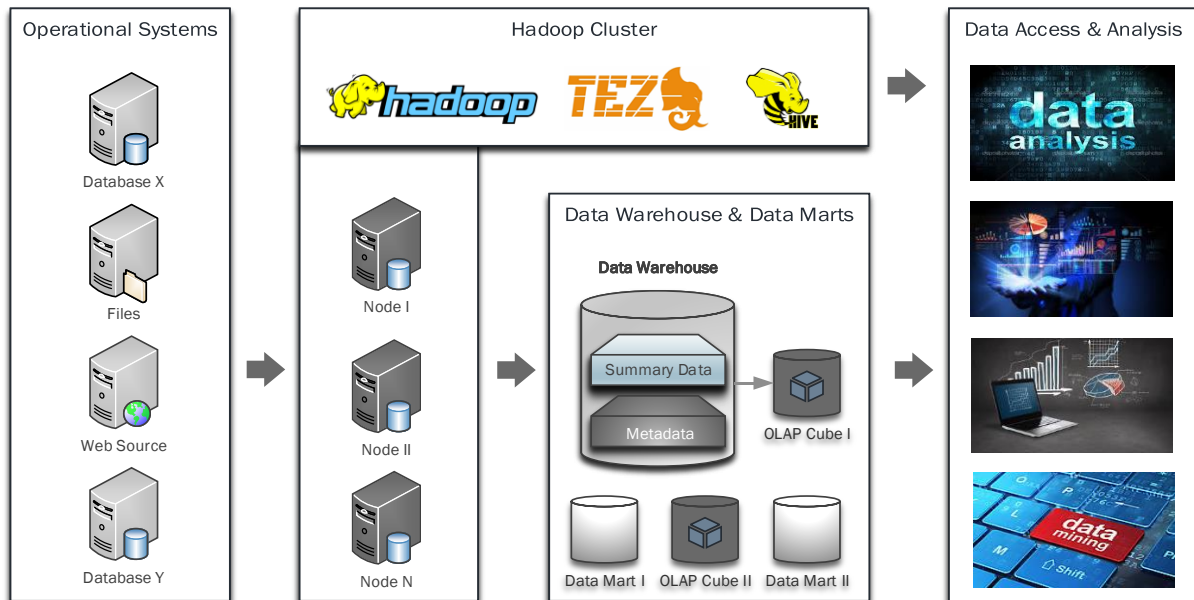


Figure 6.4. Enhanced data warehouse architecture integrating Hadoop

In Figure 6.4 we acknowledge an architecture where the ETL layer is completely replaced by a Hadoop cluster, running Hive on Tez. This cluster is responsible for the transformation and storage of detailed data, and the computation of summary data. This summary data is then integrated into the *Data Warehouse* layer, and with it there we can continue to serve the *Data Access* layer without the need for any modifications. At the same time, with Hadoop, we can store larger amounts of detailed and historical data, and this enables us to enhance the *Data Access* layer with new approaches and mechanisms in the area of Analytics.

6.6. SUMMARY

In this chapter, we analyzed some of the key aspects of the underlying architectures of RDBMSs and Hadoop. Our summarized results, regarding performance, scalability, and storage, showed us that Hadoop can greatly improve data transformations tasks but, at the same time, we verified that RDBMSs still have the upper hand when it comes to low latency and the consequent speed of interactive querying.

The results gathered from the analysis of the intrinsic characteristics of both the systems played a decisive role in the definition of the right formula, where the advantages of Big Data technologies can be used to complement RDBMSs towards a more efficient and effective data warehouse architecture.

7. CONCLUSIONS

7.1. KEY FINDINGS

Hive in conjunction with Tez, rather than with Map-Reduce, offers a very reliable and performant solution with which we can collect great benefits from the distributed processing model implemented by Hadoop. Hive also brings a familiar layer to data warehouse developers who are used to express their data access and manipulation tasks through SQL. During our research, we observed that lately SQL gained considerably more interest in the Big Data world and to attest that we can find many projects that rely on SQL as its primary language, like Hive, Impala, Drill, and Presto or even Spark that recently also started to support SQL. The SQL approach is extremely important when we analyze traditional Data Warehousing architectures that have at their core an RDBMS, since it greatly facilitates the migration of processes and data from one technology to the other.

As it was expected, we confirmed that Hadoop thrives with large volumes of data. Hadoop's batch oriented data processing model, when compared to RDBMSs, is capable of processing larger amounts of data and in a much faster way. However, on this subject, we found out that not all transformation processes extract the same benefits from distributed processing. More than related to the transformation layer of data warehouses, we observed that Hadoop, through Hive on Tez, delivers outstanding performance results associated with the analytical layer, namely in the aggregation of large data sets that generate analytical measurements.

In combining both technologies, to create an enhanced data warehouse architecture, we also need to acknowledge Hive's storage capabilities. Beyond the fact that Hadoop is designed to work with commodity hardware, and therefore being less expensive than dedicated hardware, Hive's ORC file format offers tremendous storage savings due to its columnar compression. Similar compression models, in RDBMSs, are typically associated with high-end and more expensive systems like Exadata or Teradata.

Designing a system requires that not only the present is accounted for but also the future, and for that purpose, the subject of scalability is crucial. Through our scalability tests we observed that horizontal scalability offers far greater performance improvements than vertical scalability. Moreover, the fact that horizontal scalability is easier to implement, virtually unlimited and considerably less expensive, makes it a key aspect to take into consideration in the design of systems where the volumes of data are not constant.

RDBMSs have been around for decades and they still retain much of their value. As relational semantics continue to be improved in Big Data solutions and Lambda Architectures intend to deal with the different types of data and related access needs, RDBMS are still to be reckoned with when it comes to interactive querying. We have tested Hive's LLAP, the state of the art engine designed to improve interactive querying, and its results are still very shy of what can be accomplished by an RDBMS.

Due to the already mentioned advantages of Hadoop, namely concerning performance, storage and scalability, we believe that its inclusion within a data warehouse architecture will result in great benefits that will not only enhance its current performance but will also add several new dimensions regarding data analytics, like more in-depth analyses. Data mining activities or machine learning algorithms can make use of the detailed data stored in the cluster without affecting the performance

of the visualization layer, while the latter continues to be supported by the summarized data, previously calculated by Hadoop, but made available to the RDBMS. Hadoop's advantages can also be leveraged to support the DW 2.0 architecture, for instance as an enabler of the *Archival Sector* or as the basis for the Exploration Warehouse and the Unstructured DW, both part of the *Integrated Sector*.

Wirth's law observes a real phenomenon and depicts it with humor by stating that "software is getting slower more rapidly than hardware becomes faster" (Wirth, 1995), but with the advent of Big Data and distributed processing, this difficult balance between software and hardware performances can be tackled very easily – if it is not fast enough then just distribute the work more.

7.2. RESEARCH QUESTION AND ESTABLISHED OBJECTIVES

Our driving research question was related to the initial hypothesis stating that Hadoop could be used to augment traditional data warehouses and, consequently, assure their viability even when facing huge amounts of data. This hypothesis was validated positively in a very clear way, especially by the results obtained during our evaluation of the implemented processes. To validate the initial hypothesis, our study was guided by a set of intermediary objectives²⁰ that were verified individually, but their value, towards the final goal, has a far greater importance when viewed from a cumulative perspective.

Initially, our theoretical research took us from the Data Warehousing foundations towards the SQL-like technologies of the Big Data world where we defined that, through Hive on Tez, it would be possible to easily integrate Hadoop as part of a traditional data warehouse and with that expand greatly its capabilities (O.1). Our practical journey began by the installation of a Hadoop cluster with especial focus on Hive on Tez (O.2). With this environment as support, we implemented a dimensional model, and associated transformation processes, with the purpose of extracting television audience measurements, like *Rating*, *Reach* and *Share*, from the Mediaroom platform (O.3). These processes were then used to perform a set of benchmarking tests regarding performance and storage, in comparison to a data warehouse created in an RDBMS that had been specifically implemented for the purpose. From this comparison, we were able to extract several insights regarding the architectures of each of the systems, their weaknesses and strengths, and how they could be best combined (O.4). The next step of our research took us to a fundamental aspect in Information Systems architectures, scalability. In a world where more and more data is being constantly generated, we assessed how scalability could contribute to the viability of both systems; horizontal scalability enables an almost infinite processing capability, whereas vertical scalability, typical of traditional RDBMSs, is too constrained by the hardware limits and associated high costs (O.5). Finally, the last objective (O.6), used the task of communicating the calculated television audience measurements, to ascertain the performance of Hadoop regarding interactive querying. Here it became very clear that the performance of RDBMSs, due to their low latency, greatly outperforms the one that can be obtained from Hadoop or more precisely from Hive on Tez even with the Low Latency Analytical Processing engine. This conclusion had a decisive importance in determining the final proposed data warehouse architecture that results from the combination of the strengths of both technologies.

²⁰ Our objectives are identified in section 1.2 and more specifically in Table 1.1.

7.3. MAIN CONTRIBUTIONS

Our research demonstrates empirically that indeed Hadoop can be used to augment traditional data warehouse architectures. We believe that this is a valuable asset to solution architects and managers when they need to equate the possible solutions that will help their current DW architectures expand to face the new challenges of data, with special emphasis on volume. Moving to the world of Big Data, and in particular, by the adoption of Hadoop, does not have to follow the path of disruptive innovation. Managers can avoid the big bang adoption paradigm and choose to follow a phased approach where change, rather than being transformational, is incremental. We propose that Hadoop is integrated as part of the data warehouse architecture, and the first step to achieve that would be to move out the most time-consuming processes from the current RDBMS and into the Hadoop cluster. To help with this task, our research explored the taxonomy of typical SQL statements and identified the heavy aggregations as the processes where the most benefits can be gained from processing them in Hive rather than in an RDBMS. Our vision of an augmented data warehouse, empowered by Hadoop, portrays an architecture that, even though it adds capability to the transformation layer, remains unchanged in the user-facing layer. The visualization layer is not altered but it is complemented by a new set of possibilities enabled by the capabilities of Hadoop, namely the massive storage possibilities and the power of distributed processing that can be used to conduct more in-depth analyses, dependent on large amounts of historical data.

Related to the television audience measurements, this research produced a powerful artifact that is capable of extracting valuable information from the raw data produced by Ericsson's IPTV Mediaroom platform. We modeled, designed and implemented a dimensional model, their related data transformation processes and finally created a set of reports that can be easily integrated into a broader data warehouse with the purpose of delivering pertinent television metrics like *Rating*, *Reach* and *Share*.

During our research, we had to install and configure, in a virtual setting, the two environments responsible for the roles of portraying the RDBMS and the Hadoop cluster that served as support for the practical implementations and testing. With the purpose of helping future similar installations, we created two step-by-step guides (Annex A and Annex B) regarding the installation and configuration of both systems. These guides, not only provide instructions towards the installation and configuration of the systems, but also incorporate the solutions for the problems we faced to deploy the environments successfully.

7.4. LIMITATIONS AND RECOMMENDATIONS FOR FUTURE WORKS

The greatest limitation of our research is related to the dimension of the hardware settings. Our limited resources did not allow for a better exploration of scalability and also prevented us from being able to calculate a linear relation between performance gains and scaling. This exercise would be interesting to explore so that it could provide tangible results applicable to other situations. Associated with this idea, we believe that it would be valuable to develop an artifact that could be used to assess, generically, the performance of RDBMSs and then project the required Hadoop infrastructure that would deliver the same results and also propose new configurations that could deliver the needed results. This artifact could be extremely useful in the planning of data warehouse architectures and

their underlying infrastructures. Our research explored briefly the taxonomy of SQL statements, and a broader exploration of this subject can potentially enrich and diversify the *use cases* to be added to the proposed artifact.

Our research transposed the relational model directly to a distributed architecture without exploring the advantages that could be extracted by adapting the table-based relational model to more complex data structures like, for example, Hive's *structs*. The data used in our study is a good candidate to be stored in flexible data structures since the diverse television event types each have their own specific characteristics and by storing them in tables, with fixed structures, we are making an inefficient usage of storage that consequently has adverse impacts on performance. Structure is required to support pre-defined reports but the inner-layers of data warehouse architectures, more concerned with detailed data, could potentially benefit from a more flexible data definition.

Moving SQL from an RDBMS to Hive, especially if it follows the ANSI specifications, revealed to be a fairly straightforward task that does not require much time to accomplish. Our study focused solely on the use of SQL statements to perform the data transformations, and it did not account for the necessity to port procedures or functions. It is possible to create user-defined functions in Hive, and more recently, with Hive's HPL/SQL, it is possible to wrap SQL statements with a procedural dimension that can greatly enhance and facilitate data transformations. It is also stated that HPL/SQL can execute procedures written in other languages, like Oracle's PL/SQL or Microsoft's Transact-SQL, without the need for any adaptations. Assessing to which extent this is possible, and also the performance implications of utilizing user-defined functions or HPL/SQL, presents itself as a great opportunity to deepen the knowledge regarding the SQL implementation of Hive and how it behaves with Hadoop.

Our study started by evaluating Hadoop's performance connected to the processes typically associated with the transformation layer of ETL, but it also did a brief incursion to more analytical processes, like the aggregations. It was in these aggregation processes that we were able to find outstanding performance improvements. This suggested us that Hadoop can also play an important role in the analytical layer of data warehouses and so, future investigations, related to OLAP on Hadoop, can reveal yet another way in which Hadoop can enhance Data Warehousing architectures.

As the volumes of data increased, companies were forced to upgrade their infrastructures to maintain the viability of their decision support systems. In an economic perspective, we believe that a study within organizations, with in-premises data warehouses, regarding their hardware that was deemed obsolete, can reveal a potential costless "new" infrastructure from which a Hadoop cluster can emerge with the purpose of augmenting the existing data warehouse architecture.

Information Systems are an area in a constant state of change, and thus it is recommended that the research performed here be regularly updated to include the new tendencies and technologies. If this study had happened three years ago most probably the focus would be Map-Reduce rather than Tez, and perhaps three years from now, solutions like Apache Drill or Presto are the ones where the spotlights will be. This research worked to find a solution that, by bringing together traditional RDBMSs and Hadoop, would expand the efficiency and viability of Data Warehousing architectures. However, sustaining the viability of Information Systems requires a permanent state of curiosity, criticism, creativity and innovative drive. In light of this, our final recommendation is to include these aspects in future research that will continue to help organizations transform their data into actionable knowledge.

8. BIBLIOGRAPHY

- Abbasi, A., Sarker, S., & Chiang, R. H. L. (2016). Big Data Research in Information Systems: Toward an Inclusive Research Agenda. *Journal of the Association for Information Systems*, 17(2), 1–32. <http://doi.org/10.1017/CBO9781107415324.004>
- Apache Hadoop. (2013). HDFS Architecture Guide. Retrieved February 13, 2017, from https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html
- Apache Hive. (2016). Hive HPL/SQL. Retrieved February 18, 2017, from <https://cwiki.apache.org/confluence/pages/viewpage.action?pageId=59690156>
- Architecture of Microsoft Mediaroom*. (2008). Microsoft Corporation.
- Barnes, S., Ring, T., Gallo, J., Lewis, K., Barnes, S., & Ring, T. (2016). BI Experts' Perspective: When It's Time to Hadoop. *Business Intelligence Journal*, 21(1), 32–38.
- Barton, D., & Court, D. (2012). Making Advanced Analytics Work for You. *Harvard Business Review*, 90(10), 78–83.
- Boulekrouche, B., Jabeur, N., & Alimazighi, Z. (2015). An intelligent ETL grid-based solution to enable spatial data warehouse deployment in cyber physical system context. *Procedia Computer Science*, 56(1), 111–118. <http://doi.org/10.1016/j.procs.2015.07.176>
- Breslin, M. (2004). Data Warehousing Battle of the Giants : Comparing the Basics of the Kimball and Inmon Models. *Business Intelligence Journal*, 6–20.
- Bryman, A. (2013). Social Research Strategies. In *Social Research Methods* (4th ed., pp. 18–43). Oxford: Oxford University Press.
- Carvalho, J. A. (1998). Using the Viable System Model to Describe the role of Computer- Based Systems in Organizations. In *Proceedings of the World Multiconference on Systems, Cybernetics and Informatics* (Vol. 4, pp. 497–502). Orlando, Florida, USA.
- Chandran, R. (2013). Big Data Management Platforms: Architecting Heterogeneous Solutions. *Business Intelligence Journal*, 18(4), 32–38.
- Clegg, D. (2015). Evolving Data Warehouse and BI Architectures: The Big Data Challenge. *Business Intelligence Journal*, 20(1), 19–24. <http://doi.org/101605385>
- Cloudera. (2016). New SQL Benchmarks: Apache Impala (incubating) Uniquely Delivers Analytic Database Performance. Retrieved February 19, 2017, from <http://blog.cloudera.com/blog/2016/02/new-sql-benchmarks-apache-impala-incubating-2-3-uniquely-delivers-analytic-database-performance/>
- Codd, E. F. (1969). Derivability, Redundancy and Consistency of Relations Stored in Large Data Banks. *IBM Research Report*. San Jose, California, RJ599.
- Codd, E. F. (1970). A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM*, 13(6), 377–387. <http://doi.org/10.1145/362384.362685>
- Codd, E. F. (1985). How Relational Is Your Database Management System? *Computerworld*, October 14 and 21.

- Codd, E. F., Codd, S. B., & Salley, C. T. (1993). *Providing OLAP (On-line Analytical Processing) to User-analysts: An IT Mandate*. Codd & Associates.
- Creswell, J. W. (2003). A Framework for Design. In *Research Design: Qualitative, Quantitative and Mixed Methods Approaches* (2nd ed., pp. 3–26). Thousand Oaks, CA: SAGE Publications.
- Creswell, J. W. (2013). *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches. Research design Qualitative quantitative and mixed methods approaches* (4th ed.). Thousand Oaks, CA: SAGE Publications.
- Crotty, M. (1998). *The Foundations of Social Research: Meaning and Perspective in the Research Process*. London: SAGE Publications.
- Darwen, H., & Date, C. J. (1995). The Third Manifesto. *SIGMOD Rec.*, 24(1), 39–49.
<http://doi.org/10.1145/202660.202667>
- Date, C. J. (2003). *An Introduction to Database Systems* (8th ed.). Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.
- Date, C. J. (2015). *SQL and Relational Theory: How to Write Accurate SQL Code* (3rd ed.).
<http://doi.org/10.1017/CBO9781107415324.004>
- DB-Engines. (2017). DB-Engines Ranking of Relational DBMS. Retrieved April 1, 2017, from <http://db-engines.com/en/ranking/relational+dbms>
- Dean, J., & Ghemawat, S. (2004). MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of 6th Symposium on Operating Systems Design and Implementation* (pp. 137–149). <http://doi.org/10.1145/1327452.1327492>
- Dell EMC. (2014). Data Growth, Business Opportunities, and the IT Imperatives. Retrieved May 21, 2017, from <https://www.emc.com/leadership/digital-universe/2014iview/executive-summary.htm>
- Dijcks, J., & Gubar, M. (2014). Integrating SQL and Hadoop. *Business Intelligence Journal*, 19(2), 49–55.
- Du, D. (2015). *Apache Hive Essentials*. Birmingham, UK: Packt Publishing.
- Duda, J. (2012). Business Intelligence and NoSQL Databases. *Information Systems in Management*, 1(1), 25–37.
- Dutcher, J. (2014). What Is Big Data? Retrieved June 7, 2016, from <https://datascience.berkeley.edu/what-is-big-data/>
- Easterby-Smith, M., Thorpe, R., & Jackson, P. R. (2012). *Management research : Mark Easterby-Smith, Richard Thorpe and Paul Jackson* (4th ed.). Los Angeles: SAGE Publications.
- Ellis, T. J., & Levy, Y. (2010). A Guide for Novice Researchers: Design and Development Research Methods. In *Proceedings of Informing Science & IT Education Conference (InSITE)* (pp. 107–118). Retrieved from <http://proceedings.informingscience.org/InSITE2010/InSITE10p107-118Ellis725.pdf>
- Ericsson Mediaroom. (2016). Retrieved July 7, 2016, from <https://www.ericsson.com/ourportfolio/telecom-operators/mediaroom?nav=marketcategory006>
- Ericsson Mediaroom Server Operations Help*. (2015).

- Floratou, A., Minhas, U. F., & Ozcan, F. (2014). Sql-on-hadoop: Full circle back to shared-nothing database architectures. *Proceedings of the VLDB Endowment*, 7(12), 1295–1306. <http://doi.org/10.14778/2732977.2733002>
- Franks, B. (2012). To Succeed with Big Data, Start Small. In *The Promise and Challenge of Big Data* (pp. 18–19). Harvard Business Review Publishing.
- Gates, A. (2013). The Stinger Initiative: Making Apache Hive 100 Times Faster. Retrieved April 2, 2017, from <https://hortonworks.com/blog/100x-faster-hive/>
- Ghemawat, S., Gobioff, H., & Leung, S.-T. (2003). The Google File System. *SIGOPS Oper. Syst. Rev.*, 37(5), 29–43. <http://doi.org/10.1145/1165389.945450>
- Goss, R. G., & Veeramuthu, K. (2013). Heading Towards Big Data - Building A Better Data Warehouse For More Data, More Speed, And More Users. In *ASMC 2013 SEMI Advanced Semiconductor Manufacturing Conference* (pp. 220–225). IEEE. <http://doi.org/10.1109/ASMC.2013.6552808>
- Gregg, D. G., Kulkarni, U. R., & Vinzé, A. S. (2001). Understanding the Philosophical Underpinnings of Software Engineering Research in Information Systems. *Information Systems Frontiers*, 3(2), 169–183. <http://doi.org/10.1023/A:1011491322406>
- Grover, M., Malaska, T., Seidman, J., & Shapira, G. (2014). *Hadoop Application Architectures*.
- Haerder, T., & Reuter, A. (1983). Principles of Transaction-Oriented Database Recovery. *Computing Surveys*, 15(4), 287–317. <http://doi.org/10.1145/289.291>
- Helland, P. (2016). The Singular Success of SQL. *Communications of the ACM*, 59(8), 38–41. <http://doi.org/10.1145/2948983>
- Henry, R., & Venkatraman, S. (2015). Big Data Analytics The Next Big Learning Opportunity. *Academy of Information & Management Sciences Journal*, 18(2), 17–29.
- Hevner, A., & Chatterjee, S. (2010). *Design Research in Information Systems: Theory and Practice. Integrated Series in Information Systems* (Vol. 22). New York, NY, USA: Springer US. <http://doi.org/10.1007/978-1-4419-5653-8>
- Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design Science in Information Systems Research. *MIS Quarterly*, 28(1), 75–105. <http://doi.org/10.2307/25148625>
- Hogan, M. T., & Jovanovic, V. (2015). ETL Workflow Generation for Offloading Dormant Data From the Data Warehouse To Hadoop. *Issues in Information Systems*, 16(1), 91–101.
- Hortonworks. (2016a). Apache Hive vs Apache Impala Query Performance Comparison. Retrieved February 19, 2017, from <http://hortonworks.com/blog/apache-hive-vs-apache-impala-query-performance-comparison/>
- Hortonworks. (2016b). *Data Architecture Optimization (A Hortonworks White Paper)*.
- Hortonworks. (2017). Determining HDP Memory Configuration Settings. Retrieved May 1, 2017, from https://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.5.3/bk_command-line-installation/content/determine-hdp-memory-config.html
- Huai, Y., Chauhan, A., Gates, A., Hagleitner, G., Hanson, E. N., Malley, O. O., ... Zhang, X. (2014). Major Technical Advancements in Apache Hive. In *SIGMOD '14 - Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data* (pp. 1235–1246). New York, NY, USA: ACM. <http://doi.org/10.1145/2588555.2595630>

- Imhoff, C., Gallema, N., & Geiger, J. G. (2003). *Mastering Data Warehouse Design: Relational and Dimensional Techniques*. Wiley Publishing, Inc. Indianapolis, IN, USA: Wiley Publishing, Inc. Retrieved from <http://artemisa.unicauca.edu.co/~ecaldon/docs/bd/mastering.pdf>
- Inmon, B. (2013). Big Data Implementation vs. Data Warehousing. Retrieved February 19, 2017, from <http://www.b-eye-network.com/view/17017>
- Inmon, W. H. (2005). *Building the Data Warehouse* (4th ed.). Indianapolis, IN, USA: Wiley Publishing, Inc.
- Inmon, W. H., Strauss, D., & Neushloss, G. (2008). *DW 2.0: The Architecture for the Next Generation of Data Warehousing*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.
- Isard, M., Budiu, M., Yu, Y., Birrell, A., & Fetterly, D. (2007). Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks. *ACM SIGOPS Operating Systems Review*, 59–72. <http://doi.org/10.1145/1272998.1273005>
- Jagadish, H. V., Gehrke, J., Labrinidis, A., Papakonstantinou, Y., Patel, J. M., Ramakrishnan, R., & Shahabi, C. (2014). Big Data and Its Technical Challenges. *Communications of the ACM*, 57(7), 86–94. <http://doi.org/10.1145/2611567>
- Jin, X., Wah, B. W., Cheng, X., & Wang, Y. (2015). Significance and Challenges of Big Data Research. *Big Data Research*, 2(2), 59–64. <http://doi.org/10.1016/j.bdr.2015.01.006>
- Kacfeh Emani, C., Cullot, N., & Nicolle, C. (2015). Understandable Big Data: A survey. *Computer Science Review*, 17, 70–81. <http://doi.org/10.1016/j.cosrev.2015.05.002>
- Ketokivi, M., & Mantere, S. (2010). Two Strategies for Inductive Reasoning in Organizational Research. *Academy of Management Review*, 35(2), 315–333.
- Khan, M. A. U. D., Uddin, M. F., & Gupta, N. (2014). Seven V's of Big Data - Understanding Big Data to extract Value. In *Proceedings of 2014 Zone 1 Conference of the American Society for Engineering Education (ASEE Zone 1)*. IEEE. <http://doi.org/10.1109/ASEEZone1.2014.6820689>
- Kimball, R., & Caserta, J. (2004). *The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning Conforming, and Delivering Data*. Indianapolis, IN, USA: Wiley Publishing, Inc.
- Kimball, R., & Ross, M. (2013). *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling*. (Wiley, Ed.) (3rd ed.). Indianapolis, Indiana: John Wiley & Sons, Inc.
- Krishnan, K. (2013). *Data Warehousing in the Age of Big Data*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.
- Kromer, M. (2014). Modern Hybrid Big Data Warehouse Architectures. *Business Intelligence Journal*, 19(4), 48–55.
- Leedy, P. D., & Ormrod, J. E. (2010). What Is Research? In *Practical Research: Planning and Design* (9th ed., pp. 1–11). Upper Saddle River, NJ, USA: Merrill Publishing Company.
- Lopez, J. A. (2012). Best Practices for Turning Big Data into Big Insights. *Business Intelligence Journal*, 17(4), 17–21.
- Lopez, K., & D'Antoni, J. (2014). The Modern Data Warehouse-How Big Data Impacts Analytics Architecture. *Business Intelligence Journal*, 19(3), 8–15.

- Lycett, M. (2013). "Datafication": Making Sense of (Big) Data in a Complex World. *European Journal of Information Systems*, 22(4), 381–386. <http://doi.org/10.1057/ejis.2013.10>
- Malinowski, E., & Zimányi, E. (2007). *Advanced Data Warehouse Design. Multimedia Retrieval*. http://doi.org/10.1007/978-3-662-10876-5_5
- Maria, A., Florea, I., Diaconita, V., & Bologa, R. (2015). Data integration approaches using ETL. *Database Systems Journal*, VI(3), 19–27.
- Marín-Ortega, P. M., Dmitriyev, V., Abilov, M., & Gómez, J. M. (2014). ELTA: New Approach in Designing Business Intelligence Solutions in Era of Big Data. *Procedia Technology*, 16(0), 667–674. <http://doi.org/http://dx.doi.org/10.1016/j.protcy.2014.10.015>
- Marks, R. (2013). *The Big Opportunity: Audience Research Meets Big Data*. London. Retrieved from www.ipa.co.uk/document/The-Big-Opportunity-Full-Report
- Martin, J. (1983). *Managing the Data-Base Environment* (1st ed.). Upper Saddle River, NJ, USA: Prentice Hall PTR.
- Marz, N., & Warren, J. (2015). A new Paradigm for Big Data. In *Big Data - Principles and best practices of scalable real-time data systems* (Vol. 37, pp. 1–23). Shelter Island, NY 11964: Manning Publications. <http://doi.org/10.1073/pnas.0703993104>
- McAfee, A., & Brynjolfsson, E. (2012). Big Data: The Management Revolution. *Harvard Business Review*, 90(10), 60–68. <http://doi.org/10.1007/s12599-013-0249-5>
- Melnik, S., Gubarev, A., Long, J. J., Romer, G., Shivakumar, S., Tolton, M., & Vassilakis, T. (2010). Dremel: Interactive Analysis of Web-Scale Datasets. *Proceedings of the VLDB Endowment*, 3(1–2), 330–339. <http://doi.org/10.14778/1920841.1920886>
- Mohan, C. (2013). History Repeats Itself: Sensible and Nonsensical Aspects of the NoSQL Hoopla. *Edbt/Icdt*, (March), 1–6. <http://doi.org/10.1145/2452376.2452378>
- Moorthy, J., Lahiri, R., Biswas, N., Sanyal, D., Ranjan, J., Nanath, K., & Ghosh, P. (2015). Big Data: Prospects and Challenges. *Vikalpa*, 40(1), 74–96. <http://doi.org/10.1177/0256090915575450>
- Morgan, D. L. (2014). Pragmatism as a paradigm for social research. *Qualitative Inquiry*, 20(8), 1045–1053. <http://doi.org/10.1177/1077800413513733>
- Mytton, G., Diem, P., & Dam, P. H. van. (2016). Data Analysis. In *Media Audience Research: A Guide for Professionals* (3rd ed., pp. 227–240). London: SAGE Publications Ltd.
- Nelson, J. L., & Webster, J. G. (2016). Audience Currencies in the Age of Big Data. *International Journal on Media Management*, 18(1), 9–24. <http://doi.org/10.1080/14241277.2016.1166430>
- O'Reilly Media. (2015). *Big Data Now: 2015 Edition*. O'Reilly Media, Inc.
- Oracle Corporation. (2016). *Oracle Database SQL Language Reference, 12c Release 1 (12.1)*.
- Oracle Corporation. (2017). *Oracle Database SQL Tuning Guide, 12c Release 1 (12.1)*.
- Ordonez, C., & García-García, J. (2008). Referential integrity quality metrics. *Decision Support Systems*, 44(2), 495–508. <http://doi.org/10.1016/j.dss.2007.06.004>

- Österle, H., Becker, J., Frank, U., Hess, T., Karagiannis, D., Krcmar, H., ... Sinz, E. J. (2011). Memorandum on design-oriented information systems research. *European Journal of Information Systems*, 20(1), 7–10. <http://doi.org/10.1057/ejis.2010.55>
- Pavlo, A., Paulson, E., Rasin, A., Abadi, D. J., DeWitt, D. J., Madden, S., & Stonebraker, M. (2009). A Comparison of Approaches to Large-Scale Data Analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data* (Vol. 12, pp. 165–178). Providence, Rhode Island, USA: ACM. <http://doi.org/10.1145/1559845.1559865>
- Russom, P. (2014). Evolving Data Warehouse Architectures. *The Data Warehousing Institute (TDWI)*, (Second Quarter).
- Rutherglen, J., Wampler, D., & Capriolo, E. (2012). *Programming Hive* (1st ed.). O'Reilly Media, Inc.
- Saha, B., Shah, H., Seth, S., Vijayaraghavan, G., Murthy, A., & Curino, C. (2015). Apache Tez: A Unifying Framework for Modeling and Building Data Processing Applications. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data - SIGMOD '15* (pp. 1357–1369). New York, NY, USA: ACM. <http://doi.org/10.1145/2723372.2742790>
- Saunders, M., Lewis, P., & Thornhill, A. (2016). *Research Methods for Business Students* (7th ed.). Harlow, England: Pearson Education.
- Saunders, M., & Tosey, P. (2013). The Layers of Research Design. *Rapport*, (Winter 2012/2013), 58–59.
- Sen, A., & Sinha, A. P. (2005). Data Warehousing Methodologies. *Communications of the ACM*, 48(3), 79–84. <http://doi.org/10.1145/1047671.1047673>
- Shvachko, K., Kuang, H., Radia, S., & Chansler, R. (2010). The Hadoop Distributed File System. In *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)* (pp. 1–10). IEEE. <http://doi.org/10.1109/MSST.2010.5496972>
- Shvachko, K. V. (2010). HDFS Scalability: The limits to growth. *Login*, 35(2), 6–16. Retrieved from <https://www.usenix.org/publications/login/april-2010-volume-35-number-2/hdfs-scalability-limits-growth>
- Singh, R., & Kaur, P. J. (2016). Analyzing performance of Apache Tez and MapReduce with hadoop multinode cluster on Amazon cloud. *Journal of Big Data*, 3(1), 19. <http://doi.org/10.1186/s40537-016-0051-6>
- Stonebraker, M., Abadi, D., DeWitt, D. J., Madden, S., Paulson, E., Pavlo, A., & Rasin, A. (2010). MapReduce and Parallel DBMSs: Friends or Foes? *Commun. ACM*, 53(1), 64–71. <http://doi.org/10.1145/1629175.1629197>
- Šubić, T., Pošćić, P., & Jakšić, D. (2015). Big Data in Data Warehouses. In *INFuture Conference Proceedings* (pp. 235–244). <http://doi.org/10.17234/INFUTURE.2015.27>
- Sumathi, S., & Esakkirajan, S. (2007). *Fundamentals of Relational Database Management Systems (Studies in Computational Intelligence)*. New York, NY, USA: Springer-Verlag Berlin Heidelberg.
- Webster, J. G., Phalen, P. F., & Lichty, L. W. (2014). *Ratings Analysis: Audience Measurement and Analytics* (4th ed.). New York, NY, USA: Routledge. <http://doi.org/10.4324/9780203112359>
- Weinberg, P. N., Groff, J. R., & Oppel, A. J. (2010). *SQL The Complete Reference* (3rd ed.). New York, NY, USA: McGraw-Hill, Inc.

- White, T. (2015). *Hadoop: The Definitive Guide* (4th ed.). Sebastopol, CA: O'Reilly Media, Inc.
- Williams, C. (2007). Research Methods. *Journal of Business & Economic Research*, 5(3), 65–72.
<http://doi.org/10.1093/fampract/cmi221>
- Wirth, N. (1995). A Plea for Lean Software. *Computer*, 28(2), 64–68. <http://doi.org/10.1109/2.348001>
- Yahoo! (n.d.). Yahoo! Hadoop Tutorial. Retrieved February 15, 2017, from
<https://developer.yahoo.com/hadoop/tutorial/module4.html#dataflow>
- Ziora, A. C. L. (2015). The Role of Big Data Solutions in the Management of Organizations. Review of Selected Practical Examples. *Procedia Computer Science*, 65(Iccmit), 1006–1012.
<http://doi.org/10.1016/j.procs.2015.09.059>

9. APPENDIX

Appendix A. MEDIAROOM EVENT TYPES

This appendix details the information contained in each of the Mediaroom event types. Only the relevant fields for this study are presented and described, even though the event types contain more information.

Appendix A.1. CHANNEL TUNE

Field name	Field description
EVENT_TYPE	100
COD_DATE	Date of the event in the 'YYYYMMDD' format
SOURCE_TIMESTAMP	Timestamp of when the event happened
CLIENT_ID	Set-box identifier
CLIENT_TYPE	Type of set-top box
SERVICE_TYPE	Type of service (this information is empty for event type 100)
SERVICE_ID	Service identifier (this information is empty for event type 100)
CHANNEL_NBR	Number of the channel in the television grid (Ex.: 1)
STATION_ID	Identifier of the station: Live, Video-on-Demand (VoD), Pay-per-View (PPV), etc.
VIEW_MODE	Visualization mode: Full-screen or PiP (Picture-in-Picture) for example
DURATION	Duration of the event in seconds
ACTION_STATE	Status of the executed action (0 = failed, 1 = successful)
TUNE_ID	Unique identifier that links each <i>Channel Tune</i> event to its associated Event Type 114: <i>Program Watched</i>

Table 9.1. *Channel Tune* event information

Appendix A.2. BOX POWER

Field name	Field description
EVENT_TYPE	101
COD_DATE	Date of the event in the 'YYYYMMDD' format
SOURCE_TIMESTAMP	Timestamp of when the event happened
CLIENT_ID	Set-box identifier
CLIENT_TYPE	Type of set-top box
ACTION	Flag indicating the action performed (0 = power off, 1 = power on)
ACTION_STATE	Status of the executed action (0 = failed, 1 = successful)

Table 9.2. *Box Power* event information

Appendix A.3. TRICK STATE

Field name	Field description
EVENT_TYPE	104
COD_DATE	Date of the event in the 'YYYYMMDD' format

SOURCE_TIMESTAMP	Timestamp of when the event happened
CLIENT_ID	Set-box identifier
CLIENT_TYPE	Type of set-top box
SERVICE_TYPE	Type of service (this information is empty for event type 104)
CONTENT_ID	Content identifier (the VoD, DVR or Live content being visualized)
ACTION	Trick stated invoked by the user (Pause, Play, StepBackward, StepForward, FastForward, Rewind, Skip, Replay, ScanBackward, ScanForward, Live or Stop)
ACTION_STATE	Status of the executed action (0 = failed, 1 = successful)

Table 9.3. *Trick State* event information

Appendix A.4. PROGRAM WATCHED

Field name	Field description
EVENT_TYPE	114
COD_DATE	Date of the event in the 'YYYYMMDD' format
SOURCE_TIMESTAMP	Timestamp of when the event happened
CLIENT_ID	Set-box identifier
CLIENT_TYPE	Type of set-top box
SERVICE_TYPE	Type of service (this information is empty for event type 114)
SERVICE_ID	Service identifier (this information is empty for event type 114)
CONTENT_ID	Content identifier (the VoD, DVR or Live content being visualized)
VIEW_MODE	Visualization mode: Full-screen or PiP (Picture-in-Picture) for example
DURATION	Duration of the event in seconds
ACTION_STATE	Status of the executed action (0 = failed, 1 = successful)
TUNE_ID	Unique identifier that links each <i>Program Watched</i> event to its associated Event Type 100: <i>Channel Tune</i>

Table 9.4. *Program Watched* event information

Appendix A.5. DVR START RECORDING

Field name	Field description
EVENT_TYPE	115
COD_DATE	Date of the event in the 'YYYYMMDD' format
SOURCE_TIMESTAMP	Timestamp of when the event happened
CLIENT_ID	Set-box identifier
CLIENT_TYPE	Type of set-top box
SERVICE_TYPE	Type of service (this information is empty for event type 115)
SERVICE_ID	Service identifier (this information is empty for event type 115)
CONTENT_ID	Identifier of the program being recorded
STATION_ID	Identifier of the station being recorded
DURATION	Expected duration of the recording in seconds, based on the information in the Electronic Program Guide (EPG)

ACTION_STATE	Status of the executed action (0 = failed, 1 = successful)
DYNAMIC	Flag indicating if this is a dynamic or a manual recording
RECURRING	Flag indicating if this is a one-time or a recurring recording

Table 9.5. *DVR Start Recording* event information

Appendix A.6. DVR ABORT RECORDING

Field name	Field description
EVENT_TYPE	116
COD_DATE	Date of the event in the 'YYYYMMDD' format
SOURCE_TIMESTAMP	Timestamp of when the event happened
CLIENT_ID	Set-box identifier
CLIENT_TYPE	Type of set-top box
SERVICE_TYPE	Type of service (this information is empty for event type 116)
SERVICE_ID	Service identifier (this information is empty for event type 116)
CONTENT_ID	Identifier of the program being recorded
STATION_ID	Identifier of the station being recorded
ACTION_STATE	Status of the executed action (0 = failed, 1 = successful)

Table 9.6. *DVR Abort Recording* event information

Appendix A.7. DVR PLAYBACK RECORDING

Field name	Field description
EVENT_TYPE	117
COD_DATE	Date of the event in the 'YYYYMMDD' format
SOURCE_TIMESTAMP	Timestamp of when the event happened
CLIENT_ID	Set-box identifier
CLIENT_TYPE	Type of set-top box
SERVICE_TYPE	Type of service (this information is empty for event type 117)
SERVICE_ID	Service identifier (this information is empty for event type 117)
CONTENT_ID	Identifier of the recorded program
STATION_ID	Identifier of the recorded station
ACTION_STATE	Status of the executed action (0 = failed, 1 = successful)

Table 9.7. *DVR Playback Recording* event information

Appendix A.8. DVR SCHEDULE RECORDING

Field name	Field description
EVENT_TYPE	118
COD_DATE	Date of the event in the 'YYYYMMDD' format
SOURCE_TIMESTAMP	Timestamp of when the event happened
CLIENT_ID	Set-box identifier
CLIENT_TYPE	Type of set-top box

SERVICE_TYPE	Type of service (this information is empty for event type 118)
SERVICE_ID	Service identifier (this information is empty for event type 118)
CONTENT_ID	Identifier of the program being recorded
STATION_ID	Identifier of the station being recorded
DURATION	Expected duration of the recording in seconds, based on the information in the Electronic Program Guide (EPG)
ACTION_STATE	Status of the executed action (0 = failed, 1 = successful)
DYNAMIC	Flag indicating if this is a dynamic or a manual recording
RECURRING	Flag indicating if this is a one-time or a recurring recording
FREQUENCY	Frequency of the recording

Table 9.8. *DVR Schedule Recording* event information

Appendix A.9. DVR DELETE RECORDING

Field name	Field description
EVENT_TYPE	119
COD_DATE	Date of the event in the 'YYYYMMDD' format
SOURCE_TIMESTAMP	Timestamp of when the event happened
CLIENT_ID	Set-box identifier
CLIENT_TYPE	Type of set-top box
SERVICE_TYPE	Type of service (this information is empty for event type 119)
SERVICE_ID	Service identifier (this information is empty for event type 119)
CONTENT_ID	Identifier of the program being recorded
STATION_ID	Identifier of the station being recorded
ACTION_STATE	Status of the executed action (0 = failed, 1 = successful)
MANUAL_DELETION	Flag indicating by whom the program was deleted. '1' means that the user manually deleted the recording. '0' means that the system deleted the recording to make room for other recordings

Table 9.9. *DVR Delete Recording* event information

Appendix A.10. DVR CANCEL RECORDING

Field name	Field description
EVENT_TYPE	120
COD_DATE	Date of the event in the 'YYYYMMDD' format
SOURCE_TIMESTAMP	Timestamp of when the event happened
CLIENT_ID	Set-box identifier
CLIENT_TYPE	Type of set-top box
SERVICE_TYPE	Type of service (this information is empty for event type 120)
SERVICE_ID	Service identifier (this information is empty for event type 120)
CONTENT_ID	Identifier of the program being recorded
STATION_ID	Identifier of the station being recorded

DURATION	Expected duration of the recording in seconds, based on the information on the Electronic Program Guide (EPG)
ACTION_STATE	Status of the executed action (0 = failed, 1 = successful)
DYNAMIC	Flag indicating if this is a dynamic or a manual recording
RECURRING	Flag indicating if this is a one-time or a recurring recording
INST_OF_RECURRING	Flag indicating if the end user either canceled an episode of a series or the entire series
FREQUENCY	Frequency of the recording

Table 9.10. *DVR Cancel Recording* event information

Appendix B. DATA PREPARATION SCRIPTS

split.sh

```
gunzip -fc input/*.txt.gz | awk -f hrd.awk -v path=inter/
```

hrd.awk (used by split.sh)

```
BEGIN { FS = ";" } ;

{
    fo=path "MR_AL_"$1_"$5".txt";
    print $0 >> fo;
}
```

dup.sh

```
for f in inter/MR_AL_*.txt;
do echo "Processing $f file...";
    awk '!seen[$0]++' $f | gzip > "output/$(basename "$f").gz"
    rm $f
done
```

slice.sh

```
dir=output
for f in $dir/*.txt.gz;
do
    filename=$(basename "$f")
    extension="${filename##*}"
    filename="${filename%.*}"
    extension2="${extension%#*}"
    filename2="${filename%.*}"

    echo "Uncompressing $filename.$extension..."
    gunzip $f
    echo "Uncompressing $filename.$extension...done."
    echo ""
    echo "Slicing $filename...";
    split -a 2 --line-bytes=200M --numeric-suffixes --additional-
suffix=. $extension2 $dir/$filename final/$filename2 "_"
    echo "Slicing $filename...done.";
    echo ""
    echo "Compressing slices..."
    for t in final/*.txt;
    do
        t2=$(basename "$t")
        te="${t2##*}"
        t2="${t2%.*}"

        echo "    Compressing $t2.$te..."
        gzip $t
    done
    echo "Compressing slices...done."
    echo ""
    echo "Re-compressing $filename..."
    gzip $dir/$filename
    echo "Re-compressing $filename...done."
    echo ""
done
```


Appendix C. DATA DICTIONARY

Appendix C.1. STAGING AREA TABLES

SA_ACTIVITY_EVENTS	
Column Name	Description
COD_DATE	Day identifier in the format 'YYYYMMDD'
SOURCE_TIMESTAMP	Timestamp reporting the date and time when the event happened
STB_ID	Mediaroom set-top box identifier
STB_TYPE	Type of set-top box. This information is only sent when the set-top box is turned on or off
EVENT_TYPE	Unique identification number for each event type
CHANNEL_ID	<i>Empty</i> . Channel information needs to be obtained through the CHANNEL_NBR and the Channel Map associated to the set-top box
CHANNEL_NBR	Channel number where the user is tuned to
CONTENT_ID	Program identifier for Live TV or media descriptor for VoD
STATION_ID	Station identifier: Live, VoD, PPV, etc.
VIEW_MODE	Type of stream to which the user is connected (Full-screen or PiP) and the service type of the media (primary or secondary)
DURATION	Duration of the event in seconds
EXPIRATION_DATE	Date and time of when the user's access to the rented VoD expires
ACTION	Event Type 101: flag indicating the action performed (0 = power off, 1 = power on) Event Type 104: trick stated invoked by the user (Pause, Play, StepBackward, StepForward, FastForward, Rewind, Skip, Replay, ScanBackward, ScanForward, Live or Stop)
ACTION_TIMESTAMP	Timestamp of when the action was performed
ACTION_STATE	Status of the executed action (0 = failed, 1 = successful)
CATEGORY	Menu category identifier
APP_NAME	Name of the application launched by the user
MENU_ID	Menu identifier
RESOLUTION	Content resolution
CULTURE	Culture of the media (default is 'en-US')
DYNAMIC	Flag indicating if this is a dynamic or a manual recording
RECURRING	Flag indicating if this is a one-time or a recurring recording
INSTANCE_OF_RECURRING	Flag indicating if the end user either canceled an episode of a series or the entire series
FREQUENCY	Frequency of the recording
MANUAL_DELETION	Flag indicating by whom the program was deleted. '1' means that the user manually deleted the recording. '0' means that the system deleted the recording to make room for other recordings
BYTES	<i>Deprecated</i> . For Event Type 119 is returned always as zero
TUNE_ID	Identifier connecting the several <i>Program Watched</i> events, of a set-top box, to the corresponding <i>Channel Tune</i> event

Table 9.11. SA_ACTIVITY_EVENTS table information

SA_ACTIVITY_EVENTS – Mapping of Event Type information by column										
Column Name	100	101	104	114	115	116	117	118	119	120
COD_DATE										
SOURCE_TIMESTAMP										
STB_ID										
STB_TYPE										
EVENT_TYPE										
CHANNEL_ID										
CHANNEL_NBR										
CONTENT_ID										
STATION_ID										
VIEW_MODE										
DURATION										
EXPIRATION_DATE										
ACTION										
ACTION_TIMESTAMP										
ACTION_STATE										
CATEGORY										
APP_NAME										
MENU_ID										
RESOLUTION										
CULTURE										
DYNAMIC										
RECURRING										
INSTANCE_OF_RECURRING										
FREQUENCY										
MANUAL_DELETION										
BYTES										
TUNE_ID										

Table 9.12. SA_ACTIVITY_EVENTS information mapping by event

The table above depicts the information present in each column of the table SA_ACTIVITY_EVENTS, for each of the event types. Green indicates that the column is populated and yellow that it is empty.

SA_ASSET	
Column Name	Description
COD_DATE	Day identifier in the format 'YYYYMMDD'
ASSET_ID	Asset identifier
TITLE	Asset name
DESCRIPTION	Asset description
TYPE	Asset type (VoD, SVoD or Application)
LANGUAGE	Asset language
COUNTRY_REGION	Asset country of origin

PROVIDER_NAME	Asset provider name
GENRE	Asset genre (Comedy, Action, Fantasy, etc.)
SERVICE_COLLECTION_ID	Asset default Service Collection
DURATION	Asset duration in minutes
RELEASE_YEAR	Asset release year in the 'YYYY' format
STUDIO	Studio that released the asset
PRICE	Asset price
RATING	Asset content rating

Table 9.13. SA_ASSET table information

SA_CHANNEL_MAP	
Column Name	Description
COD_DATE	Day identifier in the format 'YYYYMMDD'
CHANNEL_MAP_ID	Channel Map identifier
DESCRIPTION	Channel Map name
FLG_DEFAULT	Flag stating if the Channel Map is the default one or not

Table 9.14. SA_CHANNEL_MAP table information

SA_GROUP	
Column Name	Description
COD_DATE	Day identifier in the format 'YYYYMMDD'
GROUP_ID	Group identifier
CHANNEL_MAP_ID	Channel Map assigned to the group
INTERNAL_ID	Internal Group identifier

Table 9.15. SA_GROUP table information

SA_PROGRAM	
Column Name	Description
COD_DATE	Day identifier in the format 'YYYYMMDD'
PROGRAM_ID	Program identifier
DESCRIPTION	Program name
SERVICE_ID	Service to which the program can be associated

Table 9.16. SA_PROGRAM table information

SA_SERVICE	
Column Name	Description
COD_DATE	Day identifier in the format 'YYYYMMDD'
SERVICE_ID	Service identifier

DESCRIPTION	Service name
VIEW_MODE	Service visualization mode (Full-screen or PiP)
INTENT	Visualization intent mode (Full-screen or PiP)
TYPE	Service type (DVR, Live or VoD)
MULTICAST_GRP_IP_ADDR	IP Address of the multicast Acquisition Server
VIDEO_BITRATE	Video bit rate
AUDIO_BITRATE	Audio bit rate
AUDIO_CODEC	Audio codec
PROCESS_ID	Process identifier streaming the service
PROCESS_ID_CODE	Process identifier code streaming the service

Table 9.17. SA_SERVICE table information

SA_SERVICE_COLLECTION	
Column Name	Description
COD_DATE	Day identifier in the format 'YYYYMMDD'
ID	Asset identifier associated to the Service Collection
SERVICE_COLLECTION_ID	Service Collection identifier
DESCRIPTION	Service Collection name
EPG_ID	EPG identifier associated to the Service Collection

Table 9.18. SA_SERVICE_COLLECTION table information

SA_SERVICE_COLLECTION_MAP	
Column Name	Description
COD_DATE	Day identifier in the format 'YYYYMMDD'
SERVICE_ID	Service identifier
SERVICE_COLLECTION_ID	Service Collection identifier
TYPE	Type of stream (Full-screen or PiP) plus the service type of the media (primary or secondary)
SERVICE_ORDER	Priority of the Service inside the Service Collection

Table 9.19. SA_SERVICE_COLLECTION_MAP table information

SA_STB	
Column Name	Description
COD_DATE	Day identifier in the format 'YYYYMMDD'
CLIENT_ID	Set-top box identifier
EXTERNAL_ID	External set-top box identifier
SUBSCRIBER_ID	Subscriber identifier to which the set-top box is assigned to
STATUS	Set-top box status (1 if active or 0 if inactive)
VERSION	Set-top box version

Table 9.20. SA_STB table information

SA_SUBSCRIBER_GROUP_MAP	
Column Name	Description
COD_DATE	Day identifier in the format 'YYYYMMDD'
SUBSCRIBER_ID	Subscriber identifier
GROUP_ID	Group Identifier

Table 9.21. SA_SUBSCRIBER_GROUP_MAP table information

SA_TV_CHANNEL	
Column Name	Description
COD_DATE	Day identifier in the format 'YYYYMMDD'
TUNER_POSITION	Position of the TV Channel in the Channel Map
SERVICE_COLLECTION_ID	Service Collection identifier
CHANNEL_MAP_ID	Channel Map identifier

Table 9.22. SA_TV_CHANNEL table information

Appendix C.2. INVENTORY TABLES

SS_ASSET	
Column Name	Description
COD_DATE	Day identifier in the format 'YYYYMMDD'
ASSET_ID	Asset identifier
TITLE	Asset name
DESCRIPTION	Asset description
TYPE	Asset type (VoD, SVoD or Application)
LANGUAGE	Asset language
COUNTRY_REGION	Asset country of origin
PROVIDER_NAME	Asset provider name
GENRE	Asset genre (Comedy, Action, Fantasy, etc.)
SERVICE_COLLECTION_ID	Asset default Service Collection
DURATION	Asset duration in minutes
RELEASE_YEAR	Asset release year in 'YYYY' format
STUDIO	Studio that released the asset
PRICE	Asset price
RATING	Asset content rating

Table 9.23. SS_ASSET table information

SS_CHANNEL_MAP	
Column Name	Description
COD_DATE	Day identifier in the format 'YYYYMMDD'
CHANNEL_MAP_ID	Channel Map identifier
DESCRIPTION	Channel Map name
FLG_DEFAULT	Flag stating if the Channel Map is the default one or not

Table 9.24. SS_CHANNEL_MAP table information

SS_GROUP	
Column Name	Description
COD_DATE	Day identifier in the format 'YYYYMMDD'
GROUP_ID	Group identifier
CHANNEL_MAP_ID	Channel Map assigned to the group
INTERNAL_ID	Internal Group identifier

Table 9.25. SS_GROUP table information

SS_MAP_CHANNEL_MAP_SERVICE	
Column Name	Description
COD_DATE	Day identifier in the format 'YYYYMMDD'
CHANNEL_MAP_ID	Channel Map identifier
TUNER_POSITION	Position of the TV Channel in the Channel Map
SERVICE_ID	Service identifier
SERVICE_TYPE	Service type (DVR, Live or VoD)

Table 9.26. SS_MAP_CHANNEL_MAP_SERVICE table information

SS_MAP_STB_CHANNEL_MAP	
Column Name	Description
COD_DATE	Day identifier in the format 'YYYYMMDD'
STB_ID	Mediaroom set-top box identifier
CHANNEL_MAP_ID	Channel Map identifier

Table 9.27. SS_MAP_STB_CHANNEL_MAP table information

SS_PROGRAM	
Column Name	Description
COD_DATE	Day identifier in the format 'YYYYMMDD'
PROGRAM_ID	Program identifier
DESCRIPTION	Program name
SERVICE_ID	Service to which the program can be associated

Table 9.28. SS_PROGRAM table information

SS_SERVICE	
Column Name	Description
COD_DATE	Day identifier in the format 'YYYYMMDD'
SERVICE_ID	Service identifier
DESCRIPTION	Service name
VIEW_MODE	Service visualization mode (Full-screen or PiP)
INTENT	Service visualization intent mode (Full-screen or PiP). Often the same as the VIEW_MODE
TYPE	Service type (DVR, Live or VoD)
MULTICAST_GRP_IP_ADDR	IP Address of the multicast Acquisition Server
VIDEO_BITRATE	Video bit rate
AUDIO_BITRATE	Audio bit rate
AUDIO_CODEC	Audio codec
PROCESS_ID	Process identifier streaming the service
PROCESS_ID_CODE	Process identifier code streaming the service

Table 9.29. SS_SERVICE table information

SS_SERVICE_COLLECTION	
Column Name	Description
COD_DATE	Day identifier in the format 'YYYYMMDD'
SERVICE_COLLECTION_ID	Asset identifier associated to the Service Collection
EPG_ID	Service Collection identifier
DESCRIPTION	Service Collection name
ID	EPG identifier associated to the Service Collection

Table 9.30. SS_SERVICE_COLLECTION table information

SS_SERVICE_COLLECTION_MAP	
Column Name	Description
COD_DATE	Day identifier in the format 'YYYYMMDD'
SERVICE_COLLECTION_ID	Service identifier
SERVICE_ID	Service Collection identifier
TYPE	Type of stream (Full-screen or PiP) and the service type of the media (primary or secondary)
SERVICE_ORDER	Priority of the Service inside the Service Collection

Table 9.31. SS_SERVICE_COLLECTION_MAP table information

SS_STB	
Column Name	Description
COD_DATE	Day identifier in the format 'YYYYMMDD'
STB_ID	Set-top box identifier
EXTERNAL_ID	External set-top box identifier
SUBSCRIBER_ID	Subscriber identifier to which the set-top box is assigned to
STATUS	Set-top box status (1 if active or 0 if inactive)
VERSION	Set-top box version

Table 9.32. SS_STB table information

SS_STB_GROUP_MAP	
Column Name	Description
COD_DATE	Day identifier in the format 'YYYYMMDD'
STB_ID	Set-top box identifier
GROUP_ID	Group Identifier

Table 9.33. SS_STB_GROUP_MAP table information

SS_SUBSCRIBER_GROUP_MAP	
Column Name	Description
COD_DATE	Day identifier in the format 'YYYYMMDD'
SUBSCRIBER_ID	Subscriber identifier
GROUP_ID	Group Identifier

Table 9.34. SS_SUBSCRIBER_GROUP_MAP table information

SS_TV_CHANNEL	
Column Name	Description
COD_DATE	Day identifier in the format 'YYYYMMDD'
TUNER_POSITION	Position of the TV Channel in the Channel Map
CHANNEL_MAP_ID	Service Collection identifier
SERVICE_COLLECTION_ID	Channel Map identifier

Table 9.35. SS_TV_CHANNEL table information

Appendix C.3. SUPPORT TABLES

LU_DATE	
Column Name	Description
COD_DATE	Day identifier in the format 'YYYYMMDD'
DSC_DATE	Day description to be displayed in the reports
COD_WEEK	Week identifier in the format 'YYYYIW'
COD_MONTH	Month identifier in the format 'YYYYMM'
COD_QUARTER	Quarter identifier in the format 'YYYYQ'
COD_SEMESTER	Semester identifier in the format 'YYYYS'
COD_YEAR	Year identifier in the format 'YYYY'

Table 9.36. LU_DATE table information

LU_START_GP	
Column Name	Description
COD_START_GP	Identifier of the granularity period beginning, within a day, in the format 'HH24MI'
COD_GP_DURATION	Identifier of the granularity period duration (also expresses the duration itself in minutes)
DSC_START_GP	Granularity period description to be displayed in the reports (HH24:MI – HH24:MI)

Table 9.37. LU_START_GP table information

Appendix C.4. FACT TABLES

FACT_ACTIVITY_EVENTS	
Column Name	Description
COD_DATE	Day identifier in the format 'YYYYMMDD'
COD_DATE_GP	Day and granularity period identifier in the format 'YYYYMMDDHH24MI'
COD_START_GP	Granularity period identifier in the format 'HH24MI'
SOURCE_TIMESTAMP	Timestamp reporting the date and time when the event happened
STB_ID	Mediaroom set-top box identifier
STB_TYPE	Type of set-top box. This information is only sent when the set-top box is turned on or off
EVENT_TYPE	Unique identification number for each event type
SERVICE_TYPE	Service type: Live, DVR, SVoD and VoD
CHANNEL_ID	Service identifier
CHANNEL_NBR	Channel number where the user is tuned to
CONTENT_ID	Program identifier for Live TV or media descriptor for VoD
STATION_ID	Station identifier: Live, VoD, PPV, etc.
VIEW_MODE	Type of stream to which the user is connected (Full-screen or PiP) and the service type of the media (primary or secondary)

DURATION	Duration of the event in seconds
EXPIRATION_DATE	Date and time of when the user's access to the rented VoD expires
ACTION	Event Type 101: flag indicating the action performed (0 = power off, 1 = power on) Event Type 104: trick stated invoked by the user (Pause, Play, StepBackward, StepForward, FastForward, Rewind, Skip, Replay, ScanBackward, ScanForward, Live or Stop)
ACTION_TIMESTAMP	Timestamp of when the action was performed
ACTION_STATE	Status of the executed action (0 = failed, 1 = successful)
CATEGORY	Menu category identifier
APP_NAME	Name of the application launched by the user
MENU_ID	Menu identifier
RESOLUTION	Content resolution
CULTURE	Culture of the media (default is 'en-US')
DYNAMIC	Flag indicating if this is a dynamic or a manual recording
RECURRING	Flag indicating if this is a one-time or a recurring recording
INSTANCE_OF_RECURRING	Flag indicating if the end user either canceled an episode of a series or the entire series
FREQUENCY	Frequency of the recording
MANUAL_DELETION	Flag indicating by whom the program was deleted. '1' means that the user manually deleted the recording. '0' means that the system deleted the recording to make room for other recordings
BYTES	<i>Deprecated.</i> For Event Type 119 is returned always as zero
TUNE_ID	Identifier connecting the several <i>Program Watched</i> events, of a set-top box, to the corresponding <i>Channel Tune</i> event

Table 9.38. FACT_ACTIVITY_EVENTS table information

FACT_ACTIVITY_EVENTS – Mapping of Event Type information by column										
Column Name	100	101	104	114	115	116	117	118	119	120
COD_DATE										
COD_DATE_GP										
COD_START_GP										
SOURCE_TIMESTAMP										
STB_ID										
STB_TYPE										
EVENT_TYPE										
SERVICE_TYPE										
CHANNEL_ID										
CHANNEL_NBR										
CONTENT_ID										
STATION_ID										
VIEW_MODE										

DURATION										
EXPIRATION_DATE										
ACTION										
ACTION_TIMESTAMP										
ACTION_STATE										
CATEGORY										
APP_NAME										
MENU_ID										
RESOLUTION										
CULTURE										
DYNAMIC										
RECURRING										
INSTANCE_OF_RECURRING										
FREQUENCY										
MANUAL_DELETION										
BYTES										
TUNE_ID										

Table 9.39. FACT_ACTIVITY_EVENTS information mapping by event

The table above depicts the information present in each column of the table FACT_ACTIVITY_EVENTS, for each of the event types. Green indicates that the column is populated and yellow that it is empty.

FACT_EVT_SEGMENTED	
Column Name	Description
COD_DATE	Day identifier in the format 'YYYYMMDD'
COD_DATE_GP	Day and granularity period identifier in the format 'YYYYMMDDHH24MI'
COD_START_GP	Granularity period identifier in the format 'HH24MI'
COD_GP_DURATION	Identifier of the granularity period duration (also expresses the duration itself in minutes)
EVENT_TYPE	Unique identification number for each event type (this table is only produced for the events 100 and 114)
SERVICE_TYPE	Service type: Live, DVR, SVoD and VoD.
SERVICE_ID	Service identifier
CONTENT_ID	Program identifier for Live TV or media descriptor for VoD
STB_ID	Mediaroom set-top box identifier
SOURCE_TIMESTAMP	Timestamp reporting the date and time when the event happened
DURATION	Duration of the event inside the granularity period
TOTAL_DURATION	Total duration of the event throughout all the granularity periods it crosses

Table 9.40. FACT_EVT_SEGMENTED table information

Appendix C.5. AGGREGATION TABLES

AG_LIVE_RATING_DY	
Column Name	Description
COD_DATE	Day identifier in the format 'YYYYMMDD'
SERVICE_ID	Service identifier
PROGRAM_ID	Program identifier
RNK_PROGRAM	Program ranking, within the day, according to the average number of viewers
AVG_VIEWERS	Average number of viewers that watched the program throughout its broadcast time

Table 9.41. AG_LIVE_RATING_DY table information

AG_LIVE_REACH_DY	
Column Name	Description
COD_DATE	Day identifier in the format 'YYYYMMDD'
SERVICE_ID	Service identifier
RNK_SERVICE	Service ranking, within the day, according to the number of viewers
NBR_VIEWERS	Number of viewers that tuned the service at least once during the day

Table 9.42. AG_LIVE_REACH_DY table information

AG_LIVE_SHARE_GP	
Column Name	Description
COD_DATE	Day identifier in the format 'YYYYMMDD'
COD_START_GP	Granularity period identifier in the format 'HH24MI'
COD_GP_DURATION	Identifier of the granularity period duration (also expresses the duration itself in minutes)
SERVICE_ID	Service identifier
NBR_SUBSCRIBERS	Number of subscribers that were tuned in the service during the granularity period (one subscriber can have one or more set-top boxes)
NBR_VIEWERS	Number of set-top boxes that were tuned in the service during the granularity period
DUR_VIEWING	Service viewing duration during the granularity period

Table 9.43. AG_LIVE_SHARE_GP table information

Appendix D. DATA WAREHOUSE RDMBS IMPLEMENTATION

Appendix D.1. STORAGE OPTIONS ANALYSIS

Before implementing the physical model, that would support our data, we conducted a brief analysis of the data compression options available, in order to understand its advantages and explore the best option for the study at hand. Speed is the most important attribute but size is also important to consider, since many times they are related, and of course when designing a data warehouse, planning for the storage requirements is fundamental.

The version of Oracle, being used as the RDBMS, supports basic and advance compression and, therefore, our analysis focuses on the use of both these options, plus the use of no compression, and assesses not only the performance of writing into tables, using the different compression options, but also the potential performance overhead of reading from them.

The first test comprises of loading and transforming a compressed text file of 32MB (its uncompressed size is 200MB) containing *Channel Tune* events (section 4.5.6.1).

Compression	Execution time (s)	Size (MB)	Compression ratio
None	15	248	1.0 : 1.0
Basic	16	120	2.1 : 1.0
Advanced	17	136	1.8 : 1.0

Table 9.44. Oracle 'write' compression test

From the table above we can easily state that the best compression can be obtained through the basic compression but, and due to the simplicity of the test, we do not observe large variations regarding the execution times.

The next test uses the tables generated in the previous test as input and processes its rows through the *Event Segmentation* process (section 4.5.6.3). Through this test we can observe the effect of compression for both the read and write operations.

Compression	Execution time (s)			Size (MB)	
From/To	None	Basic	Advanced	Source	Target
None	111	113	113	248	960
Basic	106	113	116	120	528
Advanced	110	114	113	136	592

Table 9.45. Oracle 'read/write' compression test

From the data in Table 9.45 we can observe again that the basic compression is able to produce smaller outputs, even when compared with the advanced compression. On the other hand, the benchmarking regarding execution time is not very conclusive nor does it point to an obvious best approach.

Based on these tests we believe that the best compression type, the one we will use for the fact tables, seems to be the *basic*. With this format, we can obtain compression ratios around 2 to 1 without any visible performance implications.

Appendix D.2. TABLESPACES

System tablespaces creation scripts

```
-- Temporary tablespace
CREATE TEMPORARY TABLESPACE TEMP TEMPFILE
  '/mnt/sdb/oradata/dw/temp01.dbf' SIZE 4G AUTOEXTEND OFF,
  '/mnt/sdb/oradata/dw/temp02.dbf' SIZE 4G AUTOEXTEND OFF,
  '/mnt/sdb/oradata/dw/temp04.dbf' SIZE 4G AUTOEXTEND OFF,
  '/mnt/sdb/oradata/dw/temp03.dbf' SIZE 4G AUTOEXTEND OFF,
  '/mnt/sdb/oradata/dw/temp05.dbf' SIZE 4G AUTOEXTEND OFF,
  '/mnt/sdb/oradata/dw/temp06.dbf' SIZE 4G AUTOEXTEND OFF
EXTENT MANAGEMENT LOCAL UNIFORM SIZE 1M;

-- Undo tablespace
CREATE UNDO TABLESPACE UNDOTBS1 DATAFILE
  '/mnt/sdb/oradata/dw/undotbs01.dbf' SIZE 1G AUTOEXTEND OFF
ONLINE
RETENTION NOGUARANTEE
BLOCKSIZE 8K
FLASHBACK ON;

-- Sysaux tablespace
CREATE TABLESPACE SYSAUX DATAFILE
  '/mnt/sdb/oradata/dw/sysaux01.dbf' SIZE 1G AUTOEXTEND ON NEXT 10M MAXSIZE UNLIMITED
LOGGING
FORCE LOGGING
DEFAULT
  NO INMEMORY
ONLINE
EXTENT MANAGEMENT LOCAL AUTOALLOCATE
BLOCKSIZE 8K
SEGMENT SPACE MANAGEMENT AUTO
FLASHBACK ON;

-- System tablespace
CREATE TABLESPACE SYSTEM DATAFILE
  '/mnt/sdb/oradata/dw/system01.dbf' SIZE 1G AUTOEXTEND ON NEXT 10M MAXSIZE UNLIMITED
LOGGING
FORCE LOGGING
DEFAULT
  NO INMEMORY
ONLINE
EXTENT MANAGEMENT LOCAL AUTOALLOCATE
BLOCKSIZE 8K
FLASHBACK ON;
```

Data tablespaces creation scripts

```
-- Tablespace for the Inventory tables
CREATE TABLESPACE HRD_DW_INV DATAFILE
  '/mnt/sdb/oradata/dw/hrd_dw_inv_01.dbf' SIZE 2G AUTOEXTEND OFF
NOLOGGING
DEFAULT
  NO INMEMORY
ONLINE
EXTENT MANAGEMENT LOCAL AUTOALLOCATE
BLOCKSIZE 8K
SEGMENT SPACE MANAGEMENT AUTO
FLASHBACK ON;

-- Tablespace for Fact tables
CREATE TABLESPACE HRD_DW_DAT DATAFILE
  '/mnt/sdb/oradata/dw/hrd_dw_dat_01.dbf' SIZE 4G AUTOEXTEND OFF,
  '/mnt/sdb/oradata/dw/hrd_dw_dat_02.dbf' SIZE 4G AUTOEXTEND OFF,
  '/mnt/sdb/oradata/dw/hrd_dw_dat_03.dbf' SIZE 4G AUTOEXTEND OFF,
```

```

    '/mnt/sdb/oradata/dw/hrd_dw_dat_04.dbf' SIZE 4G AUTOEXTEND OFF
NOLOGGING
DEFAULT
    NO INMEMORY
ONLINE
EXTENT MANAGEMENT LOCAL AUTOALLOCATE
BLOCKSIZE 8K
SEGMENT SPACE MANAGEMENT AUTO
FLASHBACK OFF;

-- Tablespace for the Segmented tale
CREATE TABLESPACE HRD_DW_SEG DATAFILE
    '/mnt/sdb/oradata/dw/hrd_dw_seg_01.dbf' SIZE 4G AUTOEXTEND OFF,
    '/mnt/sdb/oradata/dw/hrd_dw_seg_02.dbf' SIZE 4G AUTOEXTEND OFF
NOLOGGING
DEFAULT
    NO INMEMORY
ONLINE
EXTENT MANAGEMENT LOCAL AUTOALLOCATE
BLOCKSIZE 8K
SEGMENT SPACE MANAGEMENT AUTO
FLASHBACK OFF;

-- Tablespace for the Aggregation tables
CREATE TABLESPACE HRD_DW_AGG DATAFILE
    '/mnt/sdb/oradata/dw/hrd_dw_agg_01.dbf' SIZE 1G AUTOEXTEND OFF
NOLOGGING
DEFAULT
    NO INMEMORY
ONLINE
EXTENT MANAGEMENT LOCAL AUTOALLOCATE
BLOCKSIZE 8K
SEGMENT SPACE MANAGEMENT AUTO
FLASHBACK ON;

```

Appendix D.3. DIRECTORIES

```

Directories creation scripts

CREATE OR REPLACE DIRECTORY
D_BIN AS
    '/mnt/sdc/bin';

CREATE OR REPLACE DIRECTORY
D_EVT AS
    '/mnt/sdc/evt';

CREATE OR REPLACE DIRECTORY
D_INV AS
    '/mnt/sdc/inv';

CREATE OR REPLACE DIRECTORY
D_LOG AS
    '/mnt/sdc/log';

CREATE OR REPLACE DIRECTORY
D_TMP AS
    '/mnt/sdc/tmp';

```

The source files mapped to the external tables will be placed in these directories.

Appendix D.4. STAGING AREA TABLES

SA_ACTIVITY_EVENTS

```

CREATE TABLE SA_ACTIVITY_EVENTS (
  COD_DATE          NUMBER(8,0),
  SOURCE_TIMESTAMP  DATE,
  STB_ID            VARCHAR2(100),
  STB_TYPE          VARCHAR2(100),
  EVENT_TYPE        NUMBER(3,0),
  CHANNEL_ID        VARCHAR2(100),
  CHANNEL_NBR       NUMBER(6,0),
  CONTENT_ID        VARCHAR2(100),
  STATION_ID        VARCHAR2(100),
  VIEW_MODE         VARCHAR2(100),
  DURATION          NUMBER(5,0),
  EXPIRATION_DATE   DATE,
  ACTION            VARCHAR2(100),
  ACTION_TIMESTAMP  DATE,
  ACTION_STATE      VARCHAR2(100),
  CATEGORY          VARCHAR2(100),
  APP_NAME          VARCHAR2(100),
  MENU_ID           VARCHAR2(100),
  RESOLUTION        VARCHAR2(100),
  CULTURE           VARCHAR2(100),
  DYNAMIC           VARCHAR2(100),
  RECURRING         VARCHAR2(100),
  INSTANCE_OF_RECURRING VARCHAR2(100),
  FREQUENCY         VARCHAR2(100),
  MANUAL_DELETION   VARCHAR2(100),
  BYTES             NUMBER(5,0),
  TUNE_ID           VARCHAR2(100)
)
)
ORGANIZATION EXTERNAL (
  TYPE ORACLE_LOADER
  DEFAULT DIRECTORY D_EVT
  ACCESS PARAMETERS (
    RECORDS DELIMITED BY NEWLINE
    CHARACTERSET AL32UTF8
    LOGFILE D_LOG: 'sa_activity_events.log'
    BADFILE D_LOG: 'sa_activity_events.bad'
    DISCARDFILE D_LOG: 'sa_activity_events.dsc'
    PREPROCESSOR D_BIN: 'zcat'
    FIELDS TERMINATED BY ';' OPTIONALLY ENCLOSED BY '"'
    DATE_FORMAT DATE MASK "YYYY-MM-DD HH24:MI:SS"
    MISSING FIELD VALUES ARE NULL
  )
  (
    COD_DATE          CHAR(8),
    SOURCE_TIMESTAMP  DATE,
    STB_ID            CHAR(100),
    STB_TYPE          CHAR(100),
    EVENT_TYPE        CHAR(3),
    CHANNEL_ID        CHAR(100),
    CHANNEL_NBR       CHAR(6),
    CONTENT_ID        CHAR(100),
    STATION_ID        CHAR(100),
    VIEW_MODE         CHAR(100),
    DURATION          CHAR(5),
    EXPIRATION_DATE   DATE,
    ACTION            CHAR(100),
    ACTION_TIMESTAMP  DATE,
    ACTION_STATE      CHAR(100),
    CATEGORY          CHAR(100),
    APP_NAME          CHAR(100),
    MENU_ID           CHAR(100),
    RESOLUTION        CHAR(100),
    CULTURE           CHAR(100),

```



```

        DYNAMIC                CHAR(100),
        RECURRING              CHAR(100),
        INSTANCE_OF_RECURRING  CHAR(100),
        FREQUENCY              CHAR(100),
        MANUAL_DELETION        CHAR(100),
        "BYTES"                CHAR(5),
        TUNE_ID                CHAR(100)
    )
)
LOCATION ( 'MR_AL_20160504_100_00.txt.gz',
          'MR_AL_20160505_100_00.txt.gz',
          'MR_AL_20160505_100_01.txt.gz',
          'MR_AL_20160505_100_02.txt.gz',
          'MR_AL_20160505_100_03.txt.gz',
          'MR_AL_20160505_100_04.txt.gz',
          'MR_AL_20160505_100_05.txt.gz',
          'MR_AL_20160505_100_06.txt.gz',
          'MR_AL_20160505_100_07.txt.gz',
          'MR_AL_20160505_100_08.txt.gz',
          'MR_AL_20160505_100_09.txt.gz',
          'MR_AL_20160505_100_10.txt.gz',
          'MR_AL_20160505_100_11.txt.gz'
        )
)
REJECT LIMIT 100;

```

The files in the 'LOCATION' option are just an example since more files, representing the activity logs, were used for the tests.

SA_ASSET

```

CREATE TABLE SA_ASSET (
    COD_DATE          NUMBER(8,0),
    ASSET_ID          VARCHAR2(100),
    TITLE             VARCHAR2(500),
    DESCRIPTION        VARCHAR2(1024),
    TYPE              VARCHAR2(25),
    LANGUAGE           VARCHAR2(20),
    COUNTRY_REGION    VARCHAR2(20),
    PROVIDER_NAME      VARCHAR2(100),
    GENRE              VARCHAR2(100),
    SERVICE_COLLECTION_ID VARCHAR2(50),
    DURATION           NUMBER(4,0),
    RELEASE_YEAR       NUMBER(4,0),
    STUDIO            VARCHAR2(100),
    PRICE              NUMBER(5,2),
    RATING             VARCHAR2(100)
)
ORGANIZATION EXTERNAL (
    TYPE ORACLE_LOADER
    DEFAULT DIRECTORY D_INV
    ACCESS PARAMETERS (
        RECORDS DELIMITED BY NEWLINE
        CHARACTERSET AL32UTF8
        LOGFILE D_LOG: 'sa_asset.log'
        BADFILE D_LOG: 'sa_asset.bad'
        DISCARDFILE D_LOG: 'sa_asset.dsc'
        PREPROCESSOR D_BIN: 'zcat'
        FIELDS TERMINATED BY ';' OPTIONALLY ENCLOSED BY '"'
        MISSING FIELD VALUES ARE NULL
    )
    (
        COD_DATE          CHAR(8),
        SOURCE             CHAR(100),
        ASSET_ID           CHAR(100),

```

```

        TITLE                CHAR(500),
        DESCRIPTION           CHAR(1024),
        "TYPE"                CHAR(25),
        "LANGUAGE"            CHAR(20),
        COUNTRY_REGION        CHAR(20),
        PROVIDER_NAME          CHAR(100),
        GENRE                  CHAR(100),
        SERVICE_COLLECTION_ID  CHAR(50),
        DURATION               CHAR(5),
        RELEASE_YEAR           CHAR(4),
        STUDIO                 CHAR(100),
        PRICE                  CHAR(10),
        RATING                 CHAR(100)
    )
)
LOCATION ('MSTVInv_LU_ASSET_20160615.txt.gz')
)
REJECT LIMIT UNLIMITED;

```

SA_CHANNEL_MAP

```

CREATE TABLE SA_CHANNEL_MAP (
    COD_DATE          NUMBER(8,0),
    CHANNEL_MAP_ID    VARCHAR2(100),
    DESCRITPION       VARCHAR2(100),
    FLG_DEFAULT        NUMBER(1,0)
)
ORGANIZATION EXTERNAL (
    TYPE ORACLE_LOADER
    DEFAULT DIRECTORY D_INV
    ACCESS PARAMETERS (
        RECORDS DELIMITED BY NEWLINE
        CHARACTERSET AL32UTF8
        LOGFILE D_LOG: 'sa_channel_map.log'
        BADFILE D_LOG: 'sa_channel_map.bad'
        DISCARDFILE D_LOG: 'sa_channel_map.dsc'
        PREPROCESSOR D_BIN: 'zcat'
        FIELDS TERMINATED BY ';' OPTIONALLY ENCLOSED BY '"'
        MISSING FIELD VALUES ARE NULL
    )
    (
        COD_DATE          CHAR(8),
        SOURCE             CHAR(100),
        CHANNEL_MAP_ID     CHAR(100),
        DESCRITPION        CHAR(100),
        FLG_DEFAULT        CHAR(1)
    )
)
LOCATION ('MSTVInv_LU_CHANNEL_MAP_20160615.txt.gz')
)
REJECT LIMIT UNLIMITED;

```

SA_GROUP

```

CREATE TABLE SA_GROUP (
    COD_DATE          NUMBER(8,0),
    GROUP_ID          VARCHAR2(100),
    CHANNEL_MAP_ID     VARCHAR2(100),
    INTERNAL_ID        VARCHAR2(100)
)
ORGANIZATION EXTERNAL (
    TYPE ORACLE_LOADER
    DEFAULT DIRECTORY D_INV
    ACCESS PARAMETERS (
        RECORDS DELIMITED BY NEWLINE

```

```

CHARACTERSET AL32UTF8
LOGFILE D_LOG: 'sa_group.log'
BADFILE D_LOG: 'sa_group.bad'
DISCARDFILE D_LOG: 'sa_group.dsc'
PREPROCESSOR D_BIN: 'zcat'
FIELDS TERMINATED BY ';' OPTIONALLY ENCLOSED BY ''
MISSING FIELD VALUES ARE NULL
(
  COD_DATE      CHAR(8),
  SOURCE        CHAR(100),
  GROUP_ID      CHAR(100),
  CHANNEL_MAP_ID CHAR(100),
  INTERNAL_ID   CHAR(100)
)
)
LOCATION ('MSTVInv_LU_SUBS_GROUP_20160615.txt.gz')
)
REJECT LIMIT UNLIMITED;

```

SA_PROGRAM

```

CREATE TABLE SA_PROGRAM (
  COD_DATE      NUMBER(8,0),
  PROGRAM_ID    VARCHAR2(100),
  DESCRIPTION    VARCHAR2(300),
  SERVICE_ID    VARCHAR2(100)
)
ORGANIZATION EXTERNAL (
  TYPE ORACLE_LOADER
  DEFAULT DIRECTORY D_INV
  ACCESS PARAMETERS (
    RECORDS DELIMITED BY NEWLINE
    CHARACTERSET AL32UTF8
    LOGFILE D_LOG: 'sa_program.log'
    BADFILE D_LOG: 'sa_program.bad'
    DISCARDFILE D_LOG: 'sa_program.dsc'
    PREPROCESSOR D_BIN: 'zcat'
    FIELDS TERMINATED BY ';' OPTIONALLY ENCLOSED BY ''
    MISSING FIELD VALUES ARE NULL
  )
  (
    COD_DATE      CHAR(8),
    SOURCE        CHAR(100),
    PROGRAM_ID    CHAR(100),
    DESCRIPTION    CHAR(300),
    SERVICE_ID    CHAR(100)
  )
)
LOCATION ('MSTVInv_LU_PROGRAM_20160615.txt.gz')
)
REJECT LIMIT UNLIMITED;

```

SA_SERVICE

```

CREATE TABLE SA_SERVICE (
  COD_DATE      NUMBER(8,0),
  SERVICE_ID    VARCHAR2(100),
  DESCRIPTION    VARCHAR2(100),
  VIEW_MODE     VARCHAR2(100),
  INTENT        VARCHAR2(100),
  TYPE          VARCHAR2(100),
  MULTICAST_GRP_IP_ADDR VARCHAR2(100),
  VIDEO_BITRATE VARCHAR2(100),
  AUDIO_BITRATE  VARCHAR2(100),
  AUDIO_CODEC    VARCHAR2(100),

```

```

PROCESS_ID          VARCHAR2(100),
PROCESS_ID_CODE     VARCHAR2(100)
)
ORGANIZATION EXTERNAL (
  TYPE ORACLE_LOADER
  DEFAULT DIRECTORY D_INV
  ACCESS PARAMETERS (
    RECORDS DELIMITED BY NEWLINE
    CHARACTERSET AL32UTF8
    LOGFILE D_LOG: 'sa_service.log'
    BADFILE D_LOG: 'sa_service.bad'
    DISCARDFILE D_LOG: 'sa_service.dsc'
    PREPROCESSOR D_BIN: 'zcat'
    FIELDS TERMINATED BY ';' OPTIONALLY ENCLOSED BY '"'
    MISSING FIELD VALUES ARE NULL
  )
  (
    COD_DATE          CHAR(8),
    SOURCE            CHAR(100),
    SERVICE_ID        CHAR(100),
    DESCRIPTION       CHAR(100),
    VIEW_MODE         CHAR(100),
    INTENT            CHAR(100),
    TYPE              CHAR(100),
    MULTICAST_GRP_IP_ADDR CHAR(100),
    VIDEO_BITRATE     CHAR(100),
    AUDIO_BITRATE     CHAR(100),
    AUDIO_CODEC       CHAR(100),
    PROCESS_ID        CHAR(100),
    PROCESS_ID_CODE   CHAR(100)
  )
)
LOCATION ('MSTVInv_LU_SERVICE_20160615.txt.gz')
)
REJECT LIMIT UNLIMITED;

```

SA_SERVICE_COLLECTION

```

CREATE TABLE SA_SERVICE_COLLECTION (
  COD_DATE          NUMBER(8,0),
  ID                VARCHAR2(100),
  SERVICE_COLLECTION_ID VARCHAR2(100),
  DESCRIPTION       VARCHAR2(100),
  EPG_ID           VARCHAR2(100)
)
ORGANIZATION EXTERNAL (
  TYPE ORACLE_LOADER
  DEFAULT DIRECTORY D_INV
  ACCESS PARAMETERS (
    RECORDS DELIMITED BY NEWLINE
    CHARACTERSET AL32UTF8
    LOGFILE D_LOG: 'sa_service_collection.log'
    BADFILE D_LOG: 'sa_service_collection.bad'
    DISCARDFILE D_LOG: 'sa_service_collection.dsc'
    PREPROCESSOR D_BIN: 'zcat'
    FIELDS TERMINATED BY ';' OPTIONALLY ENCLOSED BY '"'
    MISSING FIELD VALUES ARE NULL
  )
  (
    COD_DATE          CHAR(8),
    SOURCE            CHAR(100),
    ID                CHAR(100),
    SERVICE_COLLECTION_ID CHAR(100),
    DESCRIPTION       CHAR(100),
    EPG_ID           CHAR(100)
  )
)
)

```

```

    LOCATION ('MSTVInv_LU_SERV_COLLECTION_20160615.txt.gz')
)
REJECT LIMIT UNLIMITED;

```

SA_SERVICE_COLLECTION_MAP

```

CREATE TABLE SA_SERVICE_COLLECTION_MAP (
  COD_DATE      NUMBER(8,0),
  SERVICE_ID     VARCHAR2(100),
  SERVICE_COLLECTION_ID VARCHAR2(100),
  TYPE          VARCHAR2(100),
  SERVICE_ORDER  NUMBER(1,0)
)
ORGANIZATION EXTERNAL (
  TYPE ORACLE_LOADER
  DEFAULT DIRECTORY D_INV
  ACCESS PARAMETERS (
    RECORDS DELIMITED BY NEWLINE
    CHARACTERSET AL32UTF8
    LOGFILE D_LOG: 'sa_service_collection_map.log'
    BADFILE D_LOG: 'sa_service_collection_map.bad'
    DISCARDFILE D_LOG: 'sa_service_collection_map.dsc'
    PREPROCESSOR D_BIN: 'zcat'
    FIELDS TERMINATED BY ';' OPTIONALLY ENCLOSED BY '"'
    MISSING FIELD VALUES ARE NULL
  )
  (
    COD_DATE      CHAR(8),
    SOURCE         CHAR(100),
    SERVICE_ID     CHAR(100),
    SERVICE_COLLECTION_ID CHAR(100),
    TYPE          CHAR(100),
    SERVICE_ORDER  CHAR(3)
  )
)
LOCATION ('MSTVInv_LU_SERV_COLL_SERV_20160615.txt.gz')
)
REJECT LIMIT UNLIMITED;

```

SA_STB

```

CREATE TABLE SA_STB (
  COD_DATE      NUMBER(8,0),
  CLIENT_ID     VARCHAR2(100),
  EXTERNAL_ID   VARCHAR2(100),
  SUBSCRIBER_ID VARCHAR2(100),
  STATUS        VARCHAR2(100),
  VERSION       VARCHAR2(100)
)
ORGANIZATION EXTERNAL (
  TYPE ORACLE_LOADER
  DEFAULT DIRECTORY D_INV
  ACCESS PARAMETERS (
    RECORDS DELIMITED BY NEWLINE
    CHARACTERSET AL32UTF8
    LOGFILE D_LOG: 'sa_stb.log'
    BADFILE D_LOG: 'sa_stb.bad'
    DISCARDFILE D_LOG: 'sa_stb.dsc'
    PREPROCESSOR D_BIN: 'zcat'
    FIELDS TERMINATED BY ';' OPTIONALLY ENCLOSED BY '"'
    MISSING FIELD VALUES ARE NULL
  )
  (
    COD_DATE      CHAR(8),
    SOURCE         CHAR(100),
    CLIENT_ID     CHAR(100),

```

```

        EXTERNAL_ID    CHAR(100),
        SUBSCRIBER_ID  CHAR(100),
        STATUS         CHAR(100),
        VERSION        CHAR(100)
    )
)
LOCATION ('MSTVInv_LU_IPTV_CLIENT_20160615.txt.gz')
)
REJECT LIMIT UNLIMITED;

```

SA_SUBSCRIBER_GROUP_MAP

```

CREATE TABLE SA_SUBSCRIBER_GROUP_MAP (
    COD_DATE          NUMBER(8,0),
    SUBSCRIBER_ID     VARCHAR2(100),
    GROUP_ID          VARCHAR2(100)
)
ORGANIZATION EXTERNAL (
    TYPE ORACLE_LOADER
    DEFAULT DIRECTORY D_INV
    ACCESS PARAMETERS (
        RECORDS DELIMITED BY NEWLINE
        CHARACTERSET AL32UTF8
        LOGFILE D_LOG:'sa_subscriber_group_map.log'
        BADFILE D_LOG:'sa_subscriber_group_map.bad'
        DISCARDFILE D_LOG:'sa_subscriber_group_map.dsc'
        PREPROCESSOR D_BIN:'zcat'
        FIELDS TERMINATED BY ';' OPTIONALLY ENCLOSED BY '"'
        MISSING FIELD VALUES ARE NULL
    )
    (
        COD_DATE          CHAR(8),
        SOURCE            CHAR(100),
        SUBSCRIBER_ID     CHAR(100),
        GROUP_ID          CHAR(100)
    )
)
LOCATION ('MSTVInv_LU_SUBS_GRP_SUBS_20160615.txt.gz')
)
REJECT LIMIT UNLIMITED;

```

SA_TV_CHANNEL

```

CREATE TABLE SA_TV_CHANNEL (
    COD_DATE          NUMBER(8,0),
    TUNER_POSITION    NUMBER(3,0),
    SERVICE_COLLECTION_ID VARCHAR2(100),
    CHANNEL_MAP_ID     VARCHAR2(100)
)
ORGANIZATION EXTERNAL (
    TYPE ORACLE_LOADER
    DEFAULT DIRECTORY D_INV
    ACCESS PARAMETERS (
        RECORDS DELIMITED BY NEWLINE
        CHARACTERSET AL32UTF8
        LOGFILE D_LOG:'sa_tv_channel.log'
        BADFILE D_LOG:'sa_tv_channel.bad'
        DISCARDFILE D_LOG:'sa_tv_channel.dsc'
        PREPROCESSOR D_BIN:'zcat'
        FIELDS TERMINATED BY ';' OPTIONALLY ENCLOSED BY '"'
        MISSING FIELD VALUES ARE NULL
    )
    (
        COD_DATE          CHAR(8),
        SOURCE            CHAR(100),
        TUNER_POSITION     CHAR(5),

```

```

        SERVICE_COLLECTION_ID CHAR(100),
        CHANNEL_MAP_ID        CHAR(100)
    )
)
LOCATION ('MSTVInv_LU_TVCHANNEL_20160615.txt.gz')
)
REJECT LIMIT UNLIMITED;

```

Gather statistics for the Staging tables script

```

BEGIN
  FOR MyTable IN (SELECT TABLE_NAME
                  FROM user_external_tables
                  ORDER BY TABLE_NAME)
  LOOP
    DBMS_STATS.GATHER_TABLE_STATS (OWNNAME      => 'HRD',
                                  TABNAME       => MyTable.TABLE_NAME,
                                  ESTIMATE_PERCENT => NULL,
                                  METHOD_OPT     => 'FOR ALL COLUMNS SIZE 1',
                                  DEGREE        => NULL,
                                  CASCADE       => TRUE,
                                  NO_INVALIDATE  => FALSE);
    DBMS_OUTPUT.PUT_LINE('Table ' || MyTable.TABLE_NAME || ' analyzed.');
```

END LOOP;

```

END;
/

```

Appendix D.5. SUPPORT TABLES

LU_DATE

```

CREATE TABLE LU_DATE (
  COD_DATE      NUMBER NOT NULL ENABLE,
  DSC_DATE      VARCHAR2(20),
  COD_WEEK      NUMBER,
  COD_MONTH     NUMBER,
  COD_QUARTER   NUMBER,
  COD_SEMESTER  NUMBER,
  COD_YEAR      NUMBER,
  CONSTRAINT PK_IOT_LU_DATE PRIMARY KEY (COD_DATE) ENABLE
)
ORGANIZATION INDEX NOCOMPRESS
TABLESPACE HRD_DW_INV;

```

LU_START_GP

```

CREATE TABLE LU_START_GP (
  COD_START_GP   NUMBER NOT NULL ENABLE,
  COD_GP_DURATION NUMBER NOT NULL ENABLE,
  DSC_START_GP   VARCHAR2(50),
  CONSTRAINT PK_START_GP PRIMARY KEY (COD_START_GP, COD_GP_DURATION) ENABLE
)
TABLESPACE HRD_DW_INV;

```

Appendix D.6. INVENTORY TABLES

SS_ASSET

```

CREATE TABLE SS_ASSET
(
  COD_DATE      NUMBER(8,0),
  ASSET_ID      VARCHAR2(100),
  TITLE         VARCHAR2(500),

```

```

DESCRIPTION          VARCHAR2(1024),
TYPE                  VARCHAR2(25),
LANGUAGE              VARCHAR2(20),
COUNTRY_REGION        VARCHAR2(20),
PROVIDER_NAME          VARCHAR2(100),
GENRE                  VARCHAR2(100),
SERVICE_COLLECTION_ID VARCHAR2(50),
DURATION               NUMBER(4,0),
RELEASE_YEAR           NUMBER(4,0),
STUDIO                 VARCHAR2(100),
PRICE                  NUMBER(5,2),
RATING                 VARCHAR2(100)
)
PARTITION BY RANGE (COD_DATE)
(PARTITION P&cod_date VALUES LESS THAN (&cod_date_next) TABLESPACE HRD_DW_INV);

```

Data loading

```

ALTER TABLE SS_ASSET TRUNCATE PARTITION P&cod_date;

INSERT /*+ APPEND */ INTO SS_ASSET PARTITION (P&cod_date)
SELECT COD_DATE,
       ASSET_ID,
       TITLE,
       DESCRIPTION,
       TYPE,
       LANGUAGE,
       COUNTRY_REGION,
       PROVIDER_NAME,
       GENRE,
       SERVICE_COLLECTION_ID,
       DURATION,
       RELEASE_YEAR,
       STUDIO,
       PRICE,
       RATING
FROM SA_ASSET
WHERE COD_DATE = &cod_date;

COMMIT;

```

SS_CHANNEL_MAP

```

CREATE TABLE SS_CHANNEL_MAP (
  COD_DATE          NUMBER(8,0),
  CHANNEL_MAP_ID    VARCHAR2(100),
  DESCRITPION       VARCHAR2(100),
  FLG_DEFAULT       NUMBER(1,0)
)
PARTITION BY RANGE (COD_DATE)
(PARTITION P&cod_date VALUES LESS THAN (&cod_date_next) TABLESPACE HRD_DW_INV);

```

Data loading

```

ALTER TABLE SS_CHANNEL_MAP TRUNCATE PARTITION P&cod_date;

INSERT /*+ APPEND */ INTO SS_CHANNEL_MAP PARTITION (P&cod_date)
SELECT COD_DATE,
       CHANNEL_MAP_ID,
       DESCRITPION,
       FLG_DEFAULT
FROM SA_CHANNEL_MAP
WHERE COD_DATE = &cod_date;

COMMIT;

```


SS_GROUP

```
CREATE TABLE SS_GROUP (  
  COD_DATE      NUMBER(8,0),  
  GROUP_ID      VARCHAR2(100),  
  CHANNEL_MAP_ID VARCHAR2(100),  
  INTERNAL_ID   VARCHAR2(100)  
)  
PARTITION BY RANGE (COD_DATE)  
(PARTITION P&cod_date VALUES LESS THAN (&cod_date_next) TABLESPACE HRD_DW_INV);
```

Data loading

```
ALTER TABLE SS_GROUP TRUNCATE PARTITION P&cod_date;  
  
INSERT /*+ APPEND */ INTO SS_GROUP PARTITION (P&cod_date)  
SELECT  
  COD_DATE,  
  GROUP_ID,  
  CHANNEL_MAP_ID,  
  INTERNAL_ID  
FROM SA_GROUP  
WHERE COD_DATE = &cod_date;  
  
COMMIT;
```

SS_PROGRAM

```
CREATE TABLE SS_PROGRAM  
(  
  COD_DATE      NUMBER(8,0),  
  PROGRAM_ID    VARCHAR2(100),  
  DESCRIPTION   VARCHAR2(300),  
  SERVICE_ID    VARCHAR2(100)  
)  
PARTITION BY RANGE (COD_DATE)  
(PARTITION P&cod_date VALUES LESS THAN (&cod_date_next) TABLESPACE HRD_DW_INV);
```

Data loading

```
ALTER TABLE SS_PROGRAM TRUNCATE PARTITION P&cod_date;  
  
INSERT /*+ APPEND */ INTO SS_PROGRAM PARTITION (P&cod_date)  
SELECT  
  COD_DATE,  
  PROGRAM_ID,  
  DESCRIPTION,  
  SERVICE_ID  
FROM SA_PROGRAM  
WHERE COD_DATE = &cod_date;  
  
-- Manually insert the 'Program Measures Totalizer' that is used to aggregate global  
-- values from all the Programs  
INSERT INTO SS_PROGRAM  
  (COD_DATE, PROGRAM_ID, DESCRIPTION, SERVICE_ID)  
VALUES  
  (&cod_date, '0', 'Program Measures Totalizer', 'GlobalIPTVTVService');  
  
COMMIT;
```

SS_SERVICE

```
CREATE TABLE SS_SERVICE (  
  COD_DATE      NUMBER(8,0),  
  SERVICE_ID    VARCHAR2(100),  
  DESCRIPTION   VARCHAR2(100),  
  VIEW_MODE     VARCHAR2(100),  
  INTENT        VARCHAR2(100),
```

```

    TYPE                VARCHAR2(100),
    MULTICAST_GRP_IP_ADDR VARCHAR2(100),
    VIDEO_BITRATE        VARCHAR2(100),
    AUDIO_BITRATE        VARCHAR2(100),
    AUDIO_CODEC          VARCHAR2(100),
    PROCESS_ID           VARCHAR2(100),
    PROCESS_ID_CODE      VARCHAR2(100)
)
PARTITION BY RANGE (COD_DATE)
(PARTITION P&cod_date VALUES LESS THAN (&cod_date_next) TABLESPACE HRD_DW_INV);

```

Data loading

```

ALTER TABLE SS_SERVICE TRUNCATE PARTITION P&cod_date;

INSERT /*+ APPEND */ INTO SS_SERVICE PARTITION (P&cod_date)
SELECT COD_DATE,
       SERVICE_ID,
       RTRIM(TRANSLATE(REGEXP_REPLACE(DESCRIPTION, '(pip)|(PIP)|(main)|(Main)', ''), '_'),
       ' '),
       VIEW_MODE,
       INTENT,
       TYPE,
       MULTICAST_GRP_IP_ADDR,
       VIDEO_BITRATE,
       AUDIO_BITRATE,
       AUDIO_CODEC,
       PROCESS_ID,
       PROCESS_ID_CODE
FROM SA_SERVICE
WHERE COD_DATE = &cod_date;

-- Manually insert the Global IPTV TV Service that is used to aggregate global values
-- from all the TV Service channels
INSERT INTO SS_SERVICE
(COD_DATE, SERVICE_ID, DESCRIPTION)
VALUES
(&cod_date, 'GlobalIPTVTVService', 'Global TV Service');

COMMIT;

```

SS_SERVICE_COLLECTION

```

CREATE TABLE SS_SERVICE_COLLECTION (
    COD_DATE                NUMBER(8,0),
    SERVICE_COLLECTION_ID   VARCHAR2(100),
    EPG_ID                 VARCHAR2(100),
    DESCRIPTION             VARCHAR2(100),
    ID                    VARCHAR2(100)
)
PARTITION BY RANGE (COD_DATE)
(PARTITION P&cod_date VALUES LESS THAN (&cod_date_next) TABLESPACE HRD_DW_INV);

```

Data loading

```

ALTER TABLE SS_SERVICE_COLLECTION TRUNCATE PARTITION P&cod_date;

INSERT /*+ APPEND */ INTO SS_SERVICE_COLLECTION PARTITION (P&cod_date)
SELECT COD_DATE,
       SERVICE_COLLECTION_ID,
       EPG_ID,
       DESCRIPTION,
       ID
FROM SA_SERVICE_COLLECTION
WHERE COD_DATE = &cod_date;

COMMIT;

```

SS_SERVICE_COLLECTION_MAP

```
CREATE TABLE SS_SERVICE_COLLECTION_MAP (  
  COD_DATE          NUMBER(8,0),  
  SERVICE_COLLECTION_ID VARCHAR2(100),  
  SERVICE_ID        VARCHAR2(100),  
  TYPE              VARCHAR2(100),  
  SERVICE_ORDER     NUMBER(1,0)  
)  
PARTITION BY RANGE (COD_DATE)  
(PARTITION P&cod_date VALUES LESS THAN (&cod_date_next) TABLESPACE HRD_DW_INV);
```

Data loading

```
ALTER TABLE SS_SERVICE_COLLECTION_MAP TRUNCATE PARTITION P&cod_date;  
  
INSERT /*+ APPEND */ INTO SS_SERVICE_COLLECTION_MAP PARTITION (P&cod_date)  
SELECT COD_DATE,  
       SERVICE_COLLECTION_ID,  
       SERVICE_ID,  
       TYPE,  
       SERVICE_ORDER  
FROM SA_SERVICE_COLLECTION_MAP  
WHERE COD_DATE = &cod_date;  
  
COMMIT;
```

SS_STB

```
CREATE TABLE SS_STB (  
  COD_DATE          NUMBER(8,0),  
  STB_ID            VARCHAR2(100),  
  EXTERNAL_ID       VARCHAR2(100),  
  SUBSCRIBER_ID     VARCHAR2(100),  
  STATUS            VARCHAR2(100),  
  VERSION           VARCHAR2(100)  
)  
PARTITION BY RANGE (COD_DATE)  
(PARTITION P&cod_date VALUES LESS THAN (&cod_date_next) TABLESPACE HRD_DW_INV);
```

Data loading

```
ALTER TABLE SS_STB TRUNCATE PARTITION P&cod_date;  
  
INSERT /*+ APPEND */ INTO SS_STB PARTITION (P&cod_date)  
SELECT COD_DATE,  
       CLIENT_ID,  
       EXTERNAL_ID,  
       SUBSCRIBER_ID,  
       STATUS,  
       VERSION  
FROM SA_STB  
WHERE COD_DATE = &cod_date;  
  
COMMIT;
```

SS_STB_GROUP_MAP

```
CREATE TABLE SS_STB_GROUP_MAP (  
  COD_DATE          NUMBER(8,0),  
  STB_ID            VARCHAR2(100),  
  GROUP_ID          VARCHAR2(100)  
)  
PARTITION BY RANGE (COD_DATE)  
(PARTITION P&cod_date VALUES LESS THAN (&cod_date_next) TABLESPACE HRD_DW_INV);
```

Data loading

```
ALTER TABLE SS_STB_GROUP_MAP TRUNCATE PARTITION P&&cod_date;

INSERT /*+ APPEND */ INTO SS_STB_GROUP_MAP PARTITION (P&&cod_date)
SELECT m.COD_DATE,
       m.SUBSCRIBER_ID STB_ID,
       m.GROUP_ID
  FROM SA_SUBSCRIBER_GROUP_MAP m
 WHERE EXISTS (SELECT 1
               FROM SS_STB s
               WHERE m.SUBSCRIBER_ID = s.STB_ID
                  AND s.COD_DATE = &&cod_date)
 AND m.COD_DATE = &&cod_date;

COMMIT;
```

SS_SUBSCRIBER_GROUP_MAP

```
CREATE TABLE SS_SUBSCRIBER_GROUP_MAP (
  COD_DATE      NUMBER(8,0),
  SUBSCRIBER_ID VARCHAR2(100),
  GROUP_ID      VARCHAR2(100)
)
PARTITION BY RANGE (COD_DATE)
(PARTITION P&cod_date VALUES LESS THAN (&cod_date_next) TABLESPACE HRD_DW_INV);
```

Data loading

```
ALTER TABLE SS_SUBSCRIBER_GROUP_MAP TRUNCATE PARTITION P&&cod_date;

INSERT /*+ APPEND */ INTO SS_SUBSCRIBER_GROUP_MAP PARTITION (P&&cod_date)
SELECT m.COD_DATE,
       m.SUBSCRIBER_ID,
       m.GROUP_ID
  FROM SA_SUBSCRIBER_GROUP_MAP m,
       (SELECT DISTINCT SUBSCRIBER_ID
        FROM SS_STB
        WHERE COD_DATE = P&&cod_date
       ) s
 WHERE m.SUBSCRIBER_ID = s.SUBSCRIBER_ID
 AND m.COD_DATE = P&&cod_date;

COMMIT;
```

SS_TV_CHANNEL

```
CREATE TABLE SS_TV_CHANNEL (
  COD_DATE          NUMBER(8,0),
  TUNER_POSITION    NUMBER(3,0),
  CHANNEL_MAP_ID    VARCHAR2(100),
  SERVICE_COLLECTION_ID VARCHAR2(100)
)
PARTITION BY RANGE (COD_DATE)
(PARTITION P&cod_date VALUES LESS THAN (&cod_date_next) TABLESPACE HRD_DW_INV);
```

Data loading

```
ALTER TABLE SS_TV_CHANNEL TRUNCATE PARTITION P&&cod_date;

INSERT /*+ APPEND */ INTO SS_TV_CHANNEL PARTITION (P&&cod_date)
SELECT COD_DATE,
       TUNER_POSITION,
       CHANNEL_MAP_ID,
       SERVICE_COLLECTION_ID
  FROM SA_TV_CHANNEL
 WHERE COD_DATE = &&cod_date;
```

```
COMMIT;
```

SS_MAP_CHANNEL_MAP_SERVICE

```
CREATE TABLE SS_MAP_CHANNEL_MAP_SERVICE
(
  COD_DATE      NUMBER(8,0),
  CHANNEL_MAP_ID VARCHAR2(100),
  TUNER_POSITION NUMBER(3,0),
  SERVICE_ID     VARCHAR2(100),
  SERVICE_TYPE   VARCHAR2(100)
)
PARTITION BY RANGE (COD_DATE)
(PARTITION P&cod_date VALUES LESS THAN (&cod_date_next) TABLESPACE HRD_DW_INV);
```

Data loading

```
ALTER TABLE SS_MAP_CHANNEL_MAP_SERVICE TRUNCATE PARTITION P&cod_date;

INSERT /*+ APPEND */ INTO SS_MAP_CHANNEL_MAP_SERVICE PARTITION (P&cod_date)
SELECT &cod_date COD_DATE,
       r.CHANNEL_MAP_ID,
       r.TUNER_POSITION,
       r.SERVICE_ID,
       r.SERVICE_TYPE
FROM (SELECT ch.CHANNEL_MAP_ID,
            ch.TUNER_POSITION,
            svc.SERVICE_ID,
            svc.TYPE SERVICE_TYPE,
            ROW_NUMBER()
              OVER (PARTITION BY ch.CHANNEL_MAP_ID, ch.TUNER_POSITION
                    ORDER BY CASE m.TYPE
                              WHEN 'FULLSCREEN_PRIMARY' THEN 1
                              WHEN 'FULLSCREEN_SECONDARY' THEN 2
                              END,
                    m.SERVICE_ORDER
              ) RN
FROM SS_TV_CHANNEL ch,
     SS_SERVICE_COLLECTION col,
     SS_SERVICE_COLLECTION_MAP m,
     SS_SERVICE svc
WHERE ch.SERVICE_COLLECTION_ID = col.ID
  AND col.SERVICE_COLLECTION_ID = m.SERVICE_COLLECTION_ID
  AND m.SERVICE_ID = svc.SERVICE_ID
  AND svc.VIEW_MODE = 'FULLSCREEN'
  AND ch.COD_DATE = &cod_date
  AND col.COD_DATE = &cod_date
  AND m.COD_DATE = &cod_date
  AND svc.COD_DATE = &cod_date
) r
WHERE r.RN = 1
ORDER BY r.CHANNEL_MAP_ID,
         r.TUNER_POSITION;

COMMIT;
```

SS_MAP_STB_CHANNEL_MAP

```
CREATE TABLE SS_MAP_STB_CHANNEL_MAP
(
  COD_DATE      NUMBER(8,0),
  STB_ID         VARCHAR2(100),
  CHANNEL_MAP_ID VARCHAR2(100)
)
```

```

PARTITION BY RANGE (COD_DATE)
(PARTITION P&cod_date VALUES LESS THAN (&cod_date_next) TABLESPACE HRD_DW_INV);
Data loading
ALTER TABLE SS_MAP_STB_CHANNEL_MAP TRUNCATE PARTITION P&cod_date;

INSERT /*+ APPEND */ INTO SS_MAP_STB_CHANNEL_MAP PARTITION (P&cod_date)
WITH w_stb
  AS (SELECT STB_ID,
            SUBSCRIBER_ID
      FROM SS_STB
     WHERE COD_DATE = &cod_date)
SELECT &cod_date COD_DATE,
      uni.STB_ID,
      COALESCE(m2.CHANNEL_MAP_ID, d.CHANNEL_MAP_ID) CHANNEL_MAP_ID
FROM w_stb uni
LEFT OUTER
JOIN (SELECT m1.STB_ID,
            MAX(m1.CHANNEL_MAP_ID) CHANNEL_MAP_ID
      FROM (SELECT stb.STB_ID,
                  COALESCE(c.CHANNEL_MAP_ID, s.CHANNEL_MAP_ID) CHANNEL_MAP_ID
            FROM w_stb stb
           LEFT OUTER
           JOIN (SELECT sgm.SUBSCRIBER_ID,
                       sgm.GROUP_ID,
                       g.CHANNEL_MAP_ID
                FROM SS_SUBSCRIBER_GROUP_MAP sgm,
                     SS_GROUP g
               WHERE sgm.GROUP_ID = g.GROUP_ID
                 AND g.CHANNEL_MAP_ID IS NOT NULL
                 AND sgm.COD_DATE = &cod_date
                 AND g.COD_DATE = &cod_date
              ) s
          ON (stb.SUBSCRIBER_ID = s.SUBSCRIBER_ID)
        LEFT OUTER
        JOIN (SELECT cgm.STB_ID,
                     cgm.GROUP_ID,
                     g.CHANNEL_MAP_ID
              FROM SS_STB_GROUP_MAP cgm,
                   SS_GROUP g
            WHERE cgm.GROUP_ID = g.GROUP_ID
              AND g.CHANNEL_MAP_ID IS NOT NULL
              AND cgm.COD_DATE = &cod_date
              AND g.COD_DATE = &cod_date
            ) c
          ON (stb.STB_ID = c.STB_ID)
        GROUP BY stb.STB_ID,
                  COALESCE(c.CHANNEL_MAP_ID, s.CHANNEL_MAP_ID)
      ) m1
  GROUP BY STB_ID
  HAVING COUNT(*) = 1 -- To prevent Mediaroom configurations that have the same
STB in more than one Channel Map
) m2
ON (uni.STB_ID = m2.STB_ID)
CROSS
JOIN (SELECT MAX(CHANNEL_MAP_ID) CHANNEL_MAP_ID
      FROM SS_CHANNEL_MAP
     WHERE FLG_DEFAULT = 1
       AND COD_DATE = &cod_date) d; -- Default channel map to STBs without Channel
Map or wrong configuration

COMMIT;

```

Appendix D.7. FACT TABLES

FACT_ACTIVITY_EVENTS

```
CREATE TABLE FACT_ACTIVITY_EVENTS
(
  COD_DATE          NUMBER(8,0),
  COD_DATE_GP       NUMBER(12,0),
  COD_START_GP      NUMBER(4,0),
  SOURCE_TIMESTAMP   DATE,
  STB_ID            VARCHAR2(100),
  STB_TYPE          VARCHAR2(30),
  EVENT_TYPE        NUMBER(3,0),
  SERVICE_TYPE      VARCHAR2(30),
  CHANNEL_ID        VARCHAR2(100),
  CHANNEL_NBR       NUMBER(6,0),
  CONTENT_ID        VARCHAR2(100),
  STATION_ID        VARCHAR2(100),
  VIEW_MODE         VARCHAR2(30),
  DURATION          NUMBER(5,0),
  EXPIRATION_DATE   DATE,
  ACTION            VARCHAR2(30),
  ACTION_TIMESTAMP  DATE,
  ACTION_STATE      NUMBER,
  CATEGORY          VARCHAR2(100),
  APP_NAME          VARCHAR2(100),
  MENU_ID           VARCHAR2(30),
  RESOLUTION        VARCHAR2(30),
  CULTURE           VARCHAR2(30),
  DYNAMIC           VARCHAR2(10),
  RECURRING        VARCHAR2(10),
  INSTANCE_OF_RECURRING VARCHAR2(10),
  FREQUENCY         VARCHAR2(10),
  MANUAL_DELETION   VARCHAR2(10),
  BYTES             NUMBER (5,0),
  TUNE_ID           VARCHAR2(100)
)
PARTITION BY RANGE (COD_DATE)
SUBPARTITION BY LIST (EVENT_TYPE)
SUBPARTITION TEMPLATE
( SUBPARTITION S100 VALUES (100)
  TABLESPACE HRD_DW_DAT ROW STORE COMPRESS BASIC,
  SUBPARTITION S101 VALUES (101)
  TABLESPACE HRD_DW_DAT ROW STORE COMPRESS BASIC,
  SUBPARTITION S104 VALUES (104)
  TABLESPACE HRD_DW_DAT ROW STORE COMPRESS BASIC,
  SUBPARTITION S114 VALUES (114)
  TABLESPACE HRD_DW_DAT ROW STORE COMPRESS BASIC,
  SUBPARTITION S115 VALUES (115)
  TABLESPACE HRD_DW_DAT ROW STORE COMPRESS BASIC,
  SUBPARTITION S116 VALUES (116)
  TABLESPACE HRD_DW_DAT ROW STORE COMPRESS BASIC,
  SUBPARTITION S117 VALUES (117)
  TABLESPACE HRD_DW_DAT ROW STORE COMPRESS BASIC,
  SUBPARTITION S118 VALUES (118)
  TABLESPACE HRD_DW_DAT ROW STORE COMPRESS BASIC,
  SUBPARTITION S119 VALUES (119)
  TABLESPACE HRD_DW_DAT ROW STORE COMPRESS BASIC,
  SUBPARTITION S120 VALUES (120)
  TABLESPACE HRD_DW_DAT ROW STORE COMPRESS BASIC,
  SUBPARTITION PDEF VALUES (DEFAULT)
  TABLESPACE HRD_DW_DAT ROW STORE COMPRESS BASIC
)
(PARTITION P&partition_date VALUES LESS THAN (&&value_date) TABLESPACE HRD_DW_DAT ROW
STORE COMPRESS BASIC)
TABLESPACE HRD_DW_DAT ROW STORE COMPRESS BASIC;
```

Data loading – Event 100

```
PROCEDURE load_event_100(p_cod_date IN NUMBER, p_cod_date_inv IN NUMBER DEFAULT NULL)
IS
    l_cod_date_inv NUMBER(8) := COALESCE(p_cod_date_inv, getInventoryDate);
BEGIN
    DBMS_OUTPUT.PUT_LINE('Using inventory date from ' || l_cod_date_inv);

    EXECUTE IMMEDIATE 'ALTER TABLE FACT_ACTIVITY_EVENTS TRUNCATE SUBPARTITION P' ||
p_cod_date || '_S100';

    INSERT /*+ APPEND */ INTO FACT_ACTIVITY_EVENTS
        (COD_DATE,
         COD_DATE_GP,
         COD_START_GP,
         SOURCE_TIMESTAMP,
         STB_ID,
         STB_TYPE,
         EVENT_TYPE,
         SERVICE_TYPE,
         CHANNEL_ID,
         CHANNEL_NBR,
         CONTENT_ID,
         STATION_ID,
         VIEW_MODE,
         DURATION,
         EXPIRATION_DATE,
         ACTION,
         ACTION_TIMESTAMP,
         ACTION_STATE,
         CATEGORY,
         APP_NAME,
         MENU_ID,
         RESOLUTION,
         CULTURE,
         DYNAMIC,
         RECURRING,
         INSTANCE_OF_RECURRING,
         FREQUENCY,
         MANUAL_DELETION,
         BYTES,
         TUNE_ID)
    SELECT /*+ NO_GATHER_OPTIMIZER_STATISTICS */
        sa.COD_DATE,
        utl.CodDateGP(utl.trunc2GP(sa.SOURCE_TIMESTAMP)),
        utl.CodStartGP(utl.trunc2GP(sa.SOURCE_TIMESTAMP)),
        sa.SOURCE_TIMESTAMP,
        sa.STB_ID,
        sa.STB_TYPE,
        sa.EVENT_TYPE,
        ----- SERVICE TYPE -----
        srv.SERVICE_TYPE,
        ----- CHANNEL ID -----
        srv.SERVICE_ID,
        ----- CHANNEL NBR -----
        sa.CHANNEL_NBR,
        ----- CONTENT ID -----
        NULL,
        ----- STATION ID -----
        sa.STATION_ID,
        ----- VIEW MODE -----
        sa.VIEW_MODE,
        ----- DURATION -----
        sa.DURATION,
        ----- EXPIRATION_DATE -----
        NULL,
```



```

----- ACTION -----
'Channel Tune',
----- ACTION TIMESTAMP -----
NULL,
----- ACTION STATE -----
sa.ACTION_STATE,
----- CATEGORY -----
NULL,
----- APP NAME -----
NULL,
----- MENU ID -----
NULL,
----- RESOLUTION -----
NULL,
----- CULTURE -----
NULL,
----- DYNAMIC -----
NULL,
----- RECURRING -----
NULL,
----- INSTANCE OF RECURRING -----
NULL,
----- FREQUENCY -----
NULL,
----- MANUAL DELETION -----
NULL,
----- BYTES -----
NULL,
----- TUNE ID -----
sa.TUNE_ID
FROM (SELECT evt.*,
             chanmap.CHANNEL_MAP_ID
       FROM SA_ACTIVITY_EVENTS evt
       LEFT OUTER
       JOIN SS_MAP_STB_CHANNEL_MAP chanmap
         ON (evt.STB_ID = chanmap.STB_ID AND
            chanmap.COD_DATE = l_cod_date_inv)
       WHERE evt.EVENT_TYPE = 100 /*Channel Tune Event*/
            AND evt.COD_DATE = p_cod_date) sa
LEFT OUTER
JOIN SS_MAP_CHANNEL_MAP_SERVICE srv
  ON (sa.CHANNEL_NBR = srv.TUNER_POSITION AND
      sa.CHANNEL_MAP_ID = srv.CHANNEL_MAP_ID AND
      srv.COD_DATE = l_cod_date_inv);

COMMIT;
END;

```

Data loading – Event 101

```

PROCEDURE load_event_101(p_cod_date IN NUMBER)
IS
BEGIN

    EXECUTE IMMEDIATE 'ALTER TABLE FACT_ACTIVITY_EVENTS TRUNCATE SUBPARTITION P' ||
p_cod_date || '_S101';

    INSERT /*+ APPEND */ INTO FACT_ACTIVITY_EVENTS
    (COD_DATE,
     COD_DATE_GP,
     COD_START_GP,
     SOURCE_TIMESTAMP,
     STB_ID,
     STB_TYPE,
     EVENT_TYPE,
     SERVICE_TYPE,
     CHANNEL_ID,

```

```

CHANNEL_NBR,
CONTENT_ID,
STATION_ID,
VIEW_MODE,
DURATION,
EXPIRATION_DATE,
ACTION,
ACTION_TIMESTAMP,
ACTION_STATE,
CATEGORY,
APP_NAME,
MENU_ID,
RESOLUTION,
CULTURE,
DYNAMIC,
RECURRING,
INSTANCE_OF_RECURRING,
FREQUENCY,
MANUAL_DELETION,
BYTES,
TUNE_ID)
SELECT /*+ NO_GATHER_OPTIMIZER_STATISTICS */
    sa.COD_DATE,
    utl.CodDateGP(utl.trunc2GP(sa.SOURCE_TIMESTAMP)),
    utl.CodStartGP(utl.trunc2GP(sa.SOURCE_TIMESTAMP)),
    sa.SOURCE_TIMESTAMP,
    sa.STB_ID,
    sa.STB_TYPE,
    sa.EVENT_TYPE,
    ----- SERVICE TYPE -----
    NULL,
    ----- CHANNEL ID -----
    NULL,
    ----- CHANNEL NBR -----
    NULL,
    ----- CONTENT ID -----
    NULL,
    ----- STATION ID -----
    NULL,
    ----- VIEW MODE -----
    NULL,
    ----- DURATION -----
    NULL,
    ----- EXPIRATION_DATE -----
    NULL,
    ----- ACTION -----
    sa.ACTION,
    ----- ACTION_TIMESTAMP -----
    sa.SOURCE_TIMESTAMP,
    ----- ACTION_STATE -----
    1,
    ----- CATEGORY -----
    NULL,
    ----- APP_NAME -----
    NULL,
    ----- MENU_ID -----
    NULL,
    ----- RESOLUTION -----
    NULL,
    ----- CULTURE -----
    NULL,
    ----- DYNAMIC -----
    NULL,
    ----- RECURRING -----
    NULL,
    ----- INSTANCE OF RECURRING -----

```

```

NULL,
----- FREQUENCY -----
NULL,
----- MANUAL DELETION -----
NULL,
----- BYTES -----
NULL,
----- TUNE ID -----
NULL
FROM SA_ACTIVITY_EVENTS sa
WHERE sa.EVENT_TYPE = 101 -- Set-top Box Power (On/Off)
AND sa.COD_DATE = p_cod_date;

COMMIT;
END;

```

Data loading – Event 104

```

PROCEDURE load_event_104(p_cod_date IN NUMBER, p_cod_date_inv IN NUMBER DEFAULT NULL)
IS
    l_cod_date_inv NUMBER(8) := COALESCE(p_cod_date_inv, getInventoryDate);
BEGIN
    DBMS_OUTPUT.PUT_LINE('Using inventory date from ' || l_cod_date_inv);

    EXECUTE IMMEDIATE 'ALTER TABLE FACT_ACTIVITY_EVENTS TRUNCATE SUBPARTITION P' ||
p_cod_date || '_S104';

    INSERT /*+ APPEND */ INTO FACT_ACTIVITY_EVENTS
(COD_DATE,
COD_DATE_GP,
COD_START_GP,
SOURCE_TIMESTAMP,
STB_ID,
STB_TYPE,
EVENT_TYPE,
SERVICE_TYPE,
CHANNEL_ID,
CHANNEL_NBR,
CONTENT_ID,
STATION_ID,
VIEW_MODE,
DURATION,
EXPIRATION_DATE,
ACTION,
ACTION_TIMESTAMP,
ACTION_STATE,
CATEGORY,
APP_NAME,
MENU_ID,
RESOLUTION,
CULTURE,
DYNAMIC,
RECURRING,
INSTANCE_OF_RECURRING,
FREQUENCY,
MANUAL_DELETION,
BYTES,
TUNE_ID)
SELECT /*+ NO_GATHER_OPTIMIZER_STATISTICS */
    sa.COD_DATE,
    utl.CodDateGP(utl.trunc2GP(sa.SOURCE_TIMESTAMP)),
    utl.CodStartGP(utl.trunc2GP(sa.SOURCE_TIMESTAMP)),
    sa.SOURCE_TIMESTAMP,
    sa.STB_ID,
    sa.STB_TYPE,
    sa.EVENT_TYPE,
    ----- SERVICE TYPE -----

```

```

CASE WHEN vod.ASSET_ID IS NOT NULL
  THEN 'VOD'
  ELSE typ.SERVICE_TYPE
END SERVICE_TYPE,
----- CHANNEL ID -----
NULL,
----- CHANNEL NBR -----
NULL,
----- CONTENT ID -----
sa.CONTENT_ID,
----- STATION ID -----
NULL,
----- VIEW MODE -----
NULL,
----- DURATION -----
NULL,
----- EXPIRATION_DATE -----
NULL,
----- ACTION -----
sa.ACTION,
----- ACTION TIMESTAMP -----
sa.SOURCE_TIMESTAMP,
----- ACTION STATE -----
1,
----- CATEGORY -----
NULL,
----- APP NAME -----
NULL,
----- MENU ID -----
NULL,
----- RESOLUTION -----
NULL,
----- CULTURE -----
NULL,
----- DYNAMIC -----
NULL,
----- RECURRING -----
NULL,
----- INSTANCE OF RECURRING -----
NULL,
----- FREQUENCY -----
NULL,
----- MANUAL DELETION -----
NULL,
----- BYTES -----
NULL,
----- TUNE ID -----
NULL
FROM SA_ACTIVITY_EVENTS sa
LEFT OUTER
JOIN SS_ASSET vod
  ON (sa.CONTENT_ID = vod.ASSET_ID AND
      vod.COD_DATE = l_cod_date_inv)
LEFT OUTER
JOIN (SELECT col.ID,
             MAX(svc.TYPE) SERVICE_TYPE
      FROM SS_SERVICE_COLLECTION col,
           SS_SERVICE_COLLECTION_MAP m,
           SS_SERVICE svc
      WHERE col.COD_DATE = l_cod_date_inv
            AND col.SERVICE_COLLECTION_ID = m.SERVICE_COLLECTION_ID
            AND m.COD_DATE = l_cod_date_inv
            AND m.SERVICE_ID = svc.SERVICE_ID
            AND svc.COD_DATE = l_cod_date_inv
            AND svc.VIEW_MODE NOT LIKE 'PIP%'
      GROUP BY col.ID

```

```

        ) typ
    ON (sa.CONTENT_ID = typ.ID)
WHERE sa.EVENT_TYPE = 104 -- Trick State
AND sa.COD_DATE = p_cod_date;

COMMIT;
END;

```

Data loading – Event 114

```

PROCEDURE load_event_114(p_cod_date IN NUMBER, p_cod_date_inv IN NUMBER DEFAULT NULL)
IS
    l_cod_date_prev NUMBER(8) := TO_CHAR(TO_DATE(p_cod_date, 'YYYYMMDD') - 1, 'YYYYMMDD');
    l_cod_date_inv   NUMBER(8) := COALESCE(p_cod_date_inv, getInventoryDate);
BEGIN
    DBMS_OUTPUT.PUT_LINE('Using inventory date from ' || l_cod_date_inv);

    EXECUTE IMMEDIATE 'ALTER TABLE FACT_ACTIVITY_EVENTS TRUNCATE SUBPARTITION P' ||
p_cod_date || '_S114';

    INSERT /*+ APPEND */ INTO FACT_ACTIVITY_EVENTS
        (COD_DATE,
         COD_DATE_GP,
         COD_START_GP,
         SOURCE_TIMESTAMP,
         STB_ID,
         STB_TYPE,
         EVENT_TYPE,
         SERVICE_TYPE,
         CHANNEL_ID,
         CHANNEL_NBR,
         CONTENT_ID,
         STATION_ID,
         VIEW_MODE,
         DURATION,
         EXPIRATION_DATE,
         ACTION,
         ACTION_TIMESTAMP,
         ACTION_STATE,
         CATEGORY,
         APP_NAME,
         MENU_ID,
         RESOLUTION,
         CULTURE,
         DYNAMIC,
         RECURRING,
         INSTANCE_OF_RECURRING,
         FREQUENCY,
         MANUAL_DELETION,
         BYTES,
         TUNE_ID)
    SELECT /*+ NO_GATHER_OPTIMIZER_STATISTICS */
        sa.COD_DATE,
        utl.CodDateGP(utl.trunc2GP(sa.SOURCE_TIMESTAMP)),
        utl.CodStartGP(utl.trunc2GP(sa.SOURCE_TIMESTAMP)),
        sa.SOURCE_TIMESTAMP,
        sa.STB_ID,
        sa.STB_TYPE,
        sa.EVENT_TYPE,
        ----- SERVICE TYPE -----
        COALESCE(fact.SERVICE_TYPE, vod.TYPE),
        ----- CHANNEL ID -----
        -- CHANNEL_ID for EVENT_TYPE 114 is the SERVICE_ID of the matching EVENT_TYPE
100    fact.SERVICE_ID,
        ----- CHANNEL NBR -----
        NULL,

```

```

----- CONTENT ID -----
sa.CONTENT_ID,
----- STATION ID -----
NULL,
----- VIEW MODE -----
COALESCE(sa.VIEW_MODE, fact.VIEW_MODE),
----- DURATION -----
sa.DURATION,
----- EXPIRATION_DATE -----
NULL,
----- ACTION -----
'Program Transition',
----- ACTION TIMESTAMP -----
NULL,
----- ACTION STATE -----
1, -- Success
----- CATEGORY -----
NULL,
----- APP NAME -----
NULL,
----- MENU ID -----
NULL,
----- RESOLUTION -----
NULL,
----- CULTURE -----
NULL,
----- DYNAMIC -----
NULL,
----- RECURRING -----
NULL,
----- INSTANCE OF RECURRING -----
NULL,
----- FREQUENCY -----
NULL,
----- MANUAL DELETION -----
NULL,
----- BYTES -----
NULL,
----- TUNE ID -----
sa.TUNE_ID
FROM SA_ACTIVITY_EVENTS sa
LEFT OUTER -- STB_ID/TUNE_ID is unique for each event 100
JOIN (SELECT STB_ID, TUNE_ID, CHANNEL_ID SERVICE_ID, SERVICE_TYPE, VIEW_MODE
      FROM FACT_ACTIVITY_EVENTS
      WHERE EVENT_TYPE = 100 -- Channel Tune
      AND COD_DATE BETWEEN l_cod_date_prev AND p_cod_date
    ) fact
ON (sa.STB_ID = fact.STB_ID AND sa.TUNE_ID = fact.TUNE_ID)
LEFT OUTER -- Joining with VoDs to get a better definition of the service type for
VoDs
JOIN SS_ASSET vod
ON (vod.COD_DATE = l_cod_date_inv AND
    sa.CONTENT_ID = vod.ASSET_ID)
WHERE sa.EVENT_TYPE = 114 -- Program Transition
AND sa.COD_DATE = p_cod_date;

COMMIT;
END;

```

Data loading – Event DVR

```

PROCEDURE load_event_dvr(p_cod_date IN NUMBER, p_cod_date_inv IN NUMBER DEFAULT NULL)
IS
    l_cod_date_inv NUMBER(8) := COALESCE(p_cod_date_inv, getInventoryDate);
BEGIN
    DBMS_OUTPUT.PUT_LINE('Using inventory date from ' || l_cod_date_inv);

```

```

EXECUTE IMMEDIATE 'ALTER TABLE FACT_ACTIVITY_EVENTS TRUNCATE SUBPARTITION P' ||
p_cod_date || '_S115';
EXECUTE IMMEDIATE 'ALTER TABLE FACT_ACTIVITY_EVENTS TRUNCATE SUBPARTITION P' ||
p_cod_date || '_S116';
EXECUTE IMMEDIATE 'ALTER TABLE FACT_ACTIVITY_EVENTS TRUNCATE SUBPARTITION P' ||
p_cod_date || '_S117';
EXECUTE IMMEDIATE 'ALTER TABLE FACT_ACTIVITY_EVENTS TRUNCATE SUBPARTITION P' ||
p_cod_date || '_S118';
EXECUTE IMMEDIATE 'ALTER TABLE FACT_ACTIVITY_EVENTS TRUNCATE SUBPARTITION P' ||
p_cod_date || '_S119';
EXECUTE IMMEDIATE 'ALTER TABLE FACT_ACTIVITY_EVENTS TRUNCATE SUBPARTITION P' ||
p_cod_date || '_S120';

INSERT /*+ APPEND */ INTO FACT_ACTIVITY_EVENTS
(COD_DATE,
 COD_DATE_GP,
 COD_START_GP,
 SOURCE_TIMESTAMP,
 STB_ID,
 STB_TYPE,
 EVENT_TYPE,
 SERVICE_TYPE,
 CHANNEL_ID,
 CHANNEL_NBR,
 CONTENT_ID,
 STATION_ID,
 VIEW_MODE,
 DURATION,
 EXPIRATION_DATE,
 ACTION,
 ACTION_TIMESTAMP,
 ACTION_STATE,
 CATEGORY,
 APP_NAME,
 MENU_ID,
 RESOLUTION,
 CULTURE,
 DYNAMIC,
 RECURRING,
 INSTANCE_OF_RECURRING,
 FREQUENCY,
 MANUAL_DELETION,
 BYTES,
 TUNE_ID)
SELECT /*+ NO_GATHER_OPTIMIZER_STATISTICS */
sa.COD_DATE,
utl.CodDateGP(utl.trunc2GP(sa.SOURCE_TIMESTAMP)),
utl.CodStartGP(utl.trunc2GP(sa.SOURCE_TIMESTAMP)),
sa.SOURCE_TIMESTAMP,
sa.STB_ID,
sa.STB_TYPE,
sa.EVENT_TYPE,
----- SERVICE TYPE -----
sm.SERVICE_TYPE,
----- CHANNEL ID -----
sm.SERVICE_ID,
----- CHANNEL NBR -----
NULL,
----- CONTENT ID -----
sa.CONTENT_ID,
----- STATION ID -----
sa.STATION_ID,
----- VIEW MODE -----
NULL,
----- DURATION -----

```

```

CASE
  WHEN sa.EVENT_TYPE IN (115, 118, 120) THEN
    sa.DURATION
END DURATION,
----- EXPIRATION_DATE -----
NULL,
----- ACTION -----
CASE
  WHEN sa.EVENT_TYPE = 115 THEN 'DVR Start Recording'
  WHEN sa.EVENT_TYPE = 116 THEN 'DVR Abort Recording'
  WHEN sa.EVENT_TYPE = 117 THEN 'DVR Playback Recording'
  WHEN sa.EVENT_TYPE = 118 THEN 'DVR Schedule Recording'
  WHEN sa.EVENT_TYPE = 119 THEN 'DVR Delete Recording'
  WHEN sa.EVENT_TYPE = 120 THEN 'DVR Cancel Recording'
END ACTION,
----- ACTION_TIMESTAMP -----
CASE
  WHEN sa.EVENT_TYPE IN (116, 118, 119, 120) THEN
    sa.ACTION_TIMESTAMP
  ELSE
    sa.SOURCE_TIMESTAMP
END ACTION_TIMESTAMP,
----- ACTION_STATE -----
1,
----- CATEGORY -----
NULL,
----- APP_NAME -----
NULL,
----- MENU_ID -----
NULL,
----- RESOLUTION -----
NULL,
----- CULTURE -----
NULL,
----- DYNAMIC -----
CASE
  WHEN sa.EVENT_TYPE IN (115, 118, 120) THEN
    sa.DYNAMIC
END DYNAMIC,
----- RECURRING -----
CASE
  WHEN sa.EVENT_TYPE IN (115, 118, 120) THEN
    sa.RECURRING
END RECURRING,
----- INSTANCE_OF_RECURRING -----
CASE
  WHEN sa.EVENT_TYPE IN (120) THEN
    sa.INSTANCE_OF_RECURRING
END INSTANCE_OF_RECURRING,
----- FREQUENCY -----
CASE
  WHEN sa.EVENT_TYPE IN (118, 120) THEN
    sa.FREQUENCY
END FREQUENCY,
----- MANUAL_DELETION -----
CASE
  WHEN sa.EVENT_TYPE IN (119) THEN
    sa.MANUAL_DELETION
END MANUAL_DELETION,
----- BYTES -----
CASE
  WHEN sa.EVENT_TYPE IN (119) THEN
    sa.BYTES
END BYTES,
----- TUNE_ID -----
NULL

```



```

FROM SA_ACTIVITY_EVENTS sa
LEFT OUTER
JOIN (SELECT s.SERVICE_ID,
            s.TYPE SERVICE_TYPE,
            e.EPG_ID
      FROM SS_SERVICE s
      INNER JOIN (SELECT col.EPG_ID,
                        MAX(svc.SERVICE_ID) SERVICE_ID
                  FROM SS_SERVICE svc,
                  SS_SERVICE_COLLECTION_MAP scm,
                  SS_SERVICE_COLLECTION col
                  WHERE svc.COD_DATE = l_cod_date_inv
                        AND svc.VIEW_MODE = 'FULLSCREEN'
                        AND svc.SERVICE_ID = scm.SERVICE_ID
                        AND scm.COD_DATE = l_cod_date_inv
                        AND scm.SERVICE_COLLECTION_ID = col.SERVICE_COLLECTION_ID
                        AND col.COD_DATE = l_cod_date_inv
                  GROUP BY col.EPG_ID) e
      ON (s.SERVICE_ID = e.SERVICE_ID)
      WHERE s.COD_DATE = l_cod_date_inv
    ) sm
ON (sa.STATION_ID = sm.EPG_ID)
WHERE sa.EVENT_TYPE IN (115, 116, 117, 118, 119, 120) -- DVR related events
AND sa.COD_DATE = p_cod_date;

COMMIT;
END;

```

FACT_EVT_SEGMENTED

```

CREATE TABLE FACT_EVT_SEGMENTED
(
  COD_DATE          NUMBER(8,0),
  COD_DATE_GP       NUMBER(12,0),
  COD_START_GP      NUMBER(4,0),
  COD_GP_DURATION   NUMBER(2,0),
  EVENT_TYPE        NUMBER(3,0),
  SERVICE_TYPE      VARCHAR2(20),
  SERVICE_ID        VARCHAR2(100),
  CONTENT_ID        VARCHAR2(100),
  STB_ID            VARCHAR2(100),
  SOURCE_TIMESTAMP  DATE,
  DURATION          NUMBER(5,0),
  TOTAL_DURATION    NUMBER(5,0)
)
PARTITION BY LIST (EVENT_TYPE)
(
  PARTITION P100 VALUES (100)
    TABLESPACE HRD_DW_SEG ROW STORE COMPRESS BASIC,
  PARTITION P114 VALUES (114)
    TABLESPACE HRD_DW_SEG ROW STORE COMPRESS BASIC,
  PARTITION PDEF VALUES (DEFAULT)
    TABLESPACE HRD_DW_SEG ROW STORE COMPRESS BASIC
)
TABLESPACE HRD_DW_SEG ROW STORE COMPRESS BASIC;

```

Data loading

```

PROCEDURE load_segmented (p_cod_date IN NUMBER, p_event_type IN NUMBER DEFAULT 100)
IS
  l_cod_date_prev NUMBER(8) := TO_CHAR(TO_DATE(p_cod_date, 'YYYYMMDD') - 1, 'YYYYMMDD');
BEGIN
  EXECUTE IMMEDIATE 'TRUNCATE TABLE FACT_EVT_SEGMENTED';

  INSERT /*+ APPEND */ INTO FACT_EVT_SEGMENTED
    ( COD_DATE,

```

```

        COD_DATE_GP,
        COD_START_GP,
        COD_GP_DURATION,
        EVENT_TYPE,
        SERVICE_TYPE,
        SERVICE_ID,
        CONTENT_ID,
        STB_ID,
        SOURCE_TIMESTAMP,
        DURATION,
        TOTAL_DURATION
    )
    SELECT /*+ NO_GATHER_OPTIMIZER_STATISTICS */ *
    FROM (SELECT gp.COD_DATE,
                gp.COD_DATE_GP,
                gp.COD_START_GP,
                gp.COD_GP_DURATION,
                evt.EVENT_TYPE,
                evt.SERVICE_TYPE,
                evt.CONTENT_ID,
                evt.CHANNEL_ID SERVICE_ID,
                evt.STB_ID,
                evt.SOURCE_TIMESTAMP,
                ROUND(LEAST(300, (MY_END_DATE - DATE_GP) * 24 * 60 * 60) - GREATEST(0,
(MY_START_DATE - DATE_GP) * 24 * 60 * 60)) DURATION,
                evt.DURATION TOTAL_DURATION
    FROM (SELECT GREATEST(ae.SOURCE_TIMESTAMP, TO_DATE(p_cod_date, 'YYYYMMDD'))
MY_START_DATE,
                LEAST(ae.SOURCE_TIMESTAMP + DURATION / 60 / 60 / 24,
TO_DATE(p_cod_date, 'YYYYMMDD') + 1) MY_END_DATE,
                TO_NUMBER(TO_CHAR(utl.trunc2GP(GREATEST(ae.SOURCE_TIMESTAMP,
TO_DATE(p_cod_date, 'YYYYMMDD'))), 'YYYYMMDDHH24MI')) MY_START_DATE_GP,
                TO_NUMBER(TO_CHAR(utl.trunc2GP(LEAST(ae.SOURCE_TIMESTAMP +
ae.DURATION / 60 / 60 / 24, TO_DATE(p_cod_date, 'YYYYMMDD') + 1)), 'YYYYMMDDHH24MI'))
MY_END_DATE_GP,
                ae.*
    FROM FACT_ACTIVITY_EVENTS ae
    WHERE COD_DATE BETWEEN l_cod_date_prev AND p_cod_date
    AND EVENT_TYPE IN (p_event_type)
    AND VIEW_MODE NOT LIKE 'PIP%'
    AND DURATION > 0
    AND ROWNUM > 0 -- Forces materialization of the inner select and
speeds up the statement
    ) evt,
    (SELECT y.COD_DATE,
            y.COD_DATE * 10000 + y.COD_START_GP COD_DATE_GP,
            y.COD_START_GP,
            y.COD_GP_DURATION,
            TO_DATE(y.COD_DATE * 10000 + y.COD_START_GP, 'YYYYMMDDHH24MI')
DATE_GP
    FROM (SELECT l_cod_date_prev COD_DATE,
                COD_START_GP,
                COD_GP_DURATION,
                ROW_NUMBER() OVER (PARTITION BY NULL ORDER BY
COD_START_GP) RN
    FROM LU_START_GP
    WHERE COD_GP_DURATION = 5
    ) y -- Previous day. We only need 3 hours of GPs because the
events are truncated to a maximum of 3 hours
    WHERE y.RN >= (1440 / y.COD_GP_DURATION) - (180 / y.COD_GP_DURATION -
1)

    UNION ALL
    SELECT t.COD_DATE,
            t.COD_DATE * 10000 + t.COD_START_GP COD_DATE_GP,
            t.COD_START_GP,
            t.COD_GP_DURATION,

```

```

TO_DATE(t.COD_DATE * 10000 + t.COD_START_GP, 'YYYYMMDDHH24MI')
DATE_GP
        FROM (SELECT p_cod_date COD_DATE,
                     COD_START_GP,
                     COD_GP_DURATION
                FROM LU_START_GP
                WHERE COD_GP_DURATION = 5
                ) t -- Current day
        ) gp
        WHERE gp.COD_DATE_GP >= evt.MY_START_DATE_GP
        AND gp.COD_DATE_GP <= evt.MY_END_DATE_GP) f
WHERE f.DURATION > 0;

COMMIT;
END;

```

Appendix D.8. AGGREGATION TABLES

AG_LIVE_RATING_DY

```

CREATE TABLE AG_LIVE_RATING_DY
(
  COD_DATE      NUMBER,
  SERVICE_ID    VARCHAR2(100),
  PROGRAM_ID    VARCHAR2(100),
  RNK_PROGRAM   NUMBER,
  AVG_VIEWERS   NUMBER
)
PARTITION BY RANGE (COD_DATE)
(
  PARTITION P&&partition_date VALUES LESS THAN (&&value_date) TABLESPACE HRD_DW_AGG
)
TABLESPACE HRD_DW_AGG;

```

Data loading

```

PROCEDURE load_ag_live_rating_dy (p_cod_date      IN NUMBER,
                                  p_cod_date_inv   IN NUMBER DEFAULT getInventoryDate,
                                  p_max_rank       IN NUMBER DEFAULT 100)
IS
BEGIN
  DBMS_OUTPUT.PUT_LINE('Using inventory date from ' || p_cod_date_inv);

  EXECUTE IMMEDIATE 'ALTER TABLE AG_LIVE_RATING_DY TRUNCATE PARTITION P' || p_cod_date;

  INSERT /*+ APPEND */
  INTO AG_LIVE_RATING_DY
  ( COD_DATE,
    SERVICE_ID,
    PROGRAM_ID,
    RNK_PROGRAM,
    AVG_VIEWERS)
  WITH dat
  AS (SELECT COD_DATE,
             CASE WHEN GRP_ID = 0 THEN SERVICE_ID ELSE 'GlobalIPTVTVService' END
SERVICE_ID,
             CASE WHEN GRP_ID = 0 THEN CONTENT_ID ELSE '0' END PROGRAM_ID,
             GRP_ID,
             ROUND(AVG(NBR_STBS)) AVG_VIEWERS
        FROM (SELECT COD_DATE,
                     COD_DATE_GP,
                     COD_START_GP,
                     COD_GP_DURATION,
                     SERVICE_ID,
                     CONTENT_ID,
                     GROUPING_ID(COD_DATE, COD_DATE_GP, COD_START_GP, COD_GP_DURATION,
SERVICE_ID, CONTENT_ID) GRP_ID,

```

```

        COALESCE(COUNT(DISTINCT STB_ID), 0) NBR_STBS
    FROM FACT_EVT_SEGMENTED
    WHERE COD_DATE = p_cod_date
        AND EVENT_TYPE = 114 -- Program Transition
        AND SERVICE_TYPE = 'LIVE' -- Just Live TV
    GROUP BY GROUPING SETS ((COD_DATE, COD_DATE_GP, COD_START_GP,
COD_GP_DURATION, SERVICE_ID, CONTENT_ID),
        (COD_DATE, COD_GP_DURATION))
    )
    GROUP BY COD_DATE,
        SERVICE_ID,
        CONTENT_ID,
        GRP_ID
    )
    SELECT /*+ NO_GATHER_OPTIMIZER_STATISTICS */ *
    FROM (SELECT f.COD_DATE,
        f.SERVICE_ID,
        f.PROGRAM_ID,
        DENSE_RANK() OVER (PARTITION BY GRP_ID ORDER BY AVG_VIEWERS DESC)
RNK_PROGRAM,
        AVG_VIEWERS
    FROM dat f
    INNER JOIN SS_SERVICE s -- Excluding non-existent TV Channels
    ON (f.SERVICE_ID = s.SERVICE_ID AND
        s.COD_DATE = p_cod_date_inv)
    INNER JOIN SS_PROGRAM p -- Excluding non-existent Programs
    ON (f.PROGRAM_ID = p.PROGRAM_ID AND
        p.COD_DATE = p_cod_date_inv)
    )
    WHERE RNK_PROGRAM <= p_max_rank
    ORDER BY AVG_VIEWERS DESC;

    COMMIT;
END;

```

AG_LIVE_REACH_DY

```

CREATE TABLE AG_LIVE_REACH_DY
(
    COD_DATE    NUMBER,
    SERVICE_ID  VARCHAR2(100),
    RNK_SERVICE NUMBER,
    NBR_VIEWERS NUMBER
)
PARTITION BY RANGE (COD_DATE)
(
    PARTITION P&partition_date VALUES LESS THAN (&value_date) TABLESPACE HRD_DW_AGG
)
TABLESPACE HRD_DW_AGG;

```

Data loading

```

PROCEDURE load_ag_live_reach_dy (p_cod_date IN NUMBER)
IS
    l_cod_date_prev NUMBER(8) := TO_CHAR(TO_DATE(p_cod_date, 'YYYYMMDD') - 1, 'YYYYMMDD');
BEGIN

    EXECUTE IMMEDIATE 'ALTER TABLE AG_LIVE_REACH_DY TRUNCATE PARTITION P' || p_cod_date;

    INSERT /*+ APPEND */
    INTO AG_LIVE_REACH_DY
    ( COD_DATE,
      SERVICE_ID,
      RNK_SERVICE,
      NBR_VIEWERS)
    SELECT /*+ NO_GATHER_OPTIMIZER_STATISTICS */

```

```

        p_cod_date COD_DATE,
        f.SERVICE_ID,
        DENSE_RANK() OVER (PARTITION BY f.GRP_ID ORDER BY f.NBR_VIEWERS DESC)
    RNK_SERVICE,
    NBR_VIEWERS
    FROM (SELECT CASE WHEN GROUPING_ID(CHANNEL_ID) > 0 THEN 'GlobalIPTVTVService' ELSE
CHANNEL_ID END SERVICE_ID,
        GROUPING_ID(CHANNEL_ID) GRP_ID,
        COUNT(DISTINCT STB_ID) NBR_VIEWERS
        FROM FACT_ACTIVITY_EVENTS
        WHERE EVENT_TYPE = 100
        AND SERVICE_TYPE = 'LIVE'
        AND ((COD_DATE = p_cod_date) OR
        (COD_DATE = l_cod_date_prev AND TRUNC(SOURCE_TIMESTAMP +
(DURATION/86400)) > TO_DATE(COD_DATE, 'YYYYMMDD'))))
        GROUP BY GROUPING SETS ((CHANNEL_ID), ())
    ) f
    ORDER BY NBR_VIEWERS DESC;

COMMIT;
END;

```

AG_LIVE_SHARE_GP

```

CREATE TABLE AG_LIVE_SHARE_GP
(
    COD_DATE          NUMBER,
    COD_START_GP      NUMBER,
    COD_GP_DURATION   NUMBER,
    SERVICE_ID        VARCHAR2(100),
    NBR_SUBSCRIBERS   NUMBER,
    NBR_VIEWERS       NUMBER,
    DUR_VIEWING       NUMBER
)
PARTITION BY RANGE (COD_DATE)
(
    PARTITION P&partition_date VALUES LESS THAN (&value_date) TABLESPACE HRD_DW_AGG
)
TABLESPACE HRD_DW_AGG;

```

Data loading

```

PROCEDURE load_ag_live_share_gp(p_cod_date IN NUMBER, p_cod_date_inv IN NUMBER DEFAULT
NULL)
IS
    l_cod_date_inv NUMBER(8) := COALESCE(p_cod_date_inv, getInventoryDate);
BEGIN
    DBMS_OUTPUT.PUT_LINE('Using inventory date from ' || l_cod_date_inv);

    EXECUTE IMMEDIATE 'ALTER TABLE AG_LIVE_SHARE_GP TRUNCATE PARTITION P' || p_cod_date;

    INSERT /*+ APPEND */
    INTO AG_LIVE_SHARE_GP
    ( COD_DATE,
      COD_START_GP,
      COD_GP_DURATION,
      SERVICE_ID,
      NBR_SUBSCRIBERS,
      NBR_VIEWERS,
      DUR_VIEWING)
    SELECT /*+ NO_GATHER_OPTIMIZER_STATISTICS */
        fact.COD_DATE,
        fact.COD_START_GP,
        fact.COD_GP_DURATION,
        DECODE(GROUPING(fact.SERVICE_ID), 0, fact.SERVICE_ID, 'GlobalIPTVTVService')
SERVICE_ID,

```

```

        COALESCE(COUNT (DISTINCT stb.SUBSCRIBER_ID), 0) NBR_SUBSCRIBERS,
        COALESCE(COUNT (DISTINCT fact.STB_ID), 0) NBR_VIEWERS,
        COALESCE(SUM(fact.DURATION), 0) DUR_VIEWING
FROM FACT_EVT_SEGMENTED fact
INNER JOIN SS_STB stb
    ON (fact.STB_ID = stb.STB_ID AND
        stb.COD_DATE = l_cod_date_inv)
WHERE fact.COD_DATE = p_cod_date
    AND fact.EVENT_TYPE = 100 /* Channel Tune Event */
    AND fact.SERVICE_TYPE = 'LIVE'
GROUP BY fact.COD_DATE,
        fact.COD_START_GP,
        fact.COD_GP_DURATION,
        ROLLUP(fact.SERVICE_ID);

COMMIT;
END;

```

Appendix D.9. SUPPORT PROCEDURES

Package UTL

```

CREATE OR REPLACE PACKAGE BODY utl
IS
-----
-- Name:          trunc2GP
-- Description: Truncs a given date to the closest Granularity Period
-- Parameters:  p_gp_duration: Granularity Period size in minutes (eg.: 5)
--              p_date:       Date to be truncated
-----
FUNCTION trunc2GP(p_date      DATE DEFAULT SYSDATE,
                 p_gp_duration NUMBER DEFAULT 5) RETURN DATE DETERMINISTIC
IS
BEGIN
    RETURN p_date - MOD(TO_NUMBER(TO_CHAR(p_date, 'SSSS')), p_gp_duration * 60) / (60 *
60 * 24);
END;

-----
-- Name:          CodDateGP
-- Description: Returns a COD_DATE_GP identifier (YYYYMMDDHH24MI) from a
--              given date
-- Parameters:  p_date: date to be formatted
-----
FUNCTION CodDateGP(p_date DATE DEFAULT SYSDATE) RETURN NUMBER RESULT_CACHE
DETERMINISTIC
IS
BEGIN
    RETURN TO_NUMBER(TO_CHAR(p_date, 'YYYYMMDDHH24MI'));
END;

-----
-- Name:          CodStartGP
-- Description: Returns a COD_START_GP identifier (HH24MI) from a given date
-- Parameters:  p_date: date to be formatted
-----
FUNCTION CodStartGP(p_date DATE DEFAULT SYSDATE) RETURN NUMBER RESULT_CACHE
DETERMINISTIC
IS
BEGIN
    RETURN TO_NUMBER(TO_CHAR(p_date, 'HH24MI'));
END;

-----
-- Name:          getInventoryDate

```

```

-- Description: Gets the last date populated in the inventory tables
-- Parameters:  (none)
-----
FUNCTION getInventoryDate RETURN NUMBER RESULT_CACHE DETERMINISTIC
IS
    l_cod_date_inv NUMBER;
BEGIN
    SELECT MAX(COD_DATE)
        INTO l_cod_date_inv
        FROM SS_CHANNEL_MAP;

    RETURN l_cod_date_inv;
END;

END utl;

```

Appendix D.10. EXECUTION PLANS

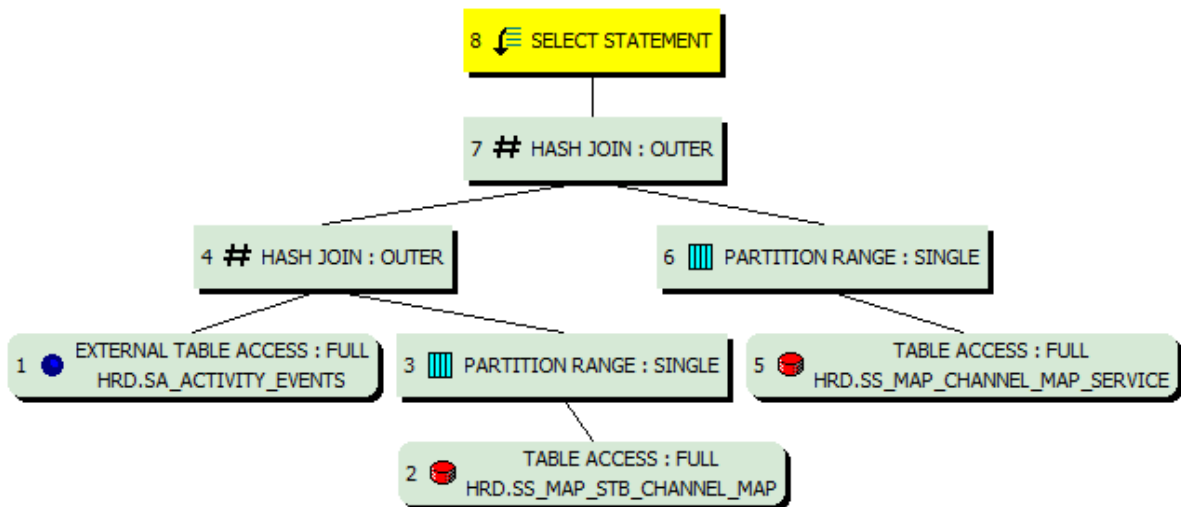


Figure 9.1. *Channel Tune* transformation execution plan

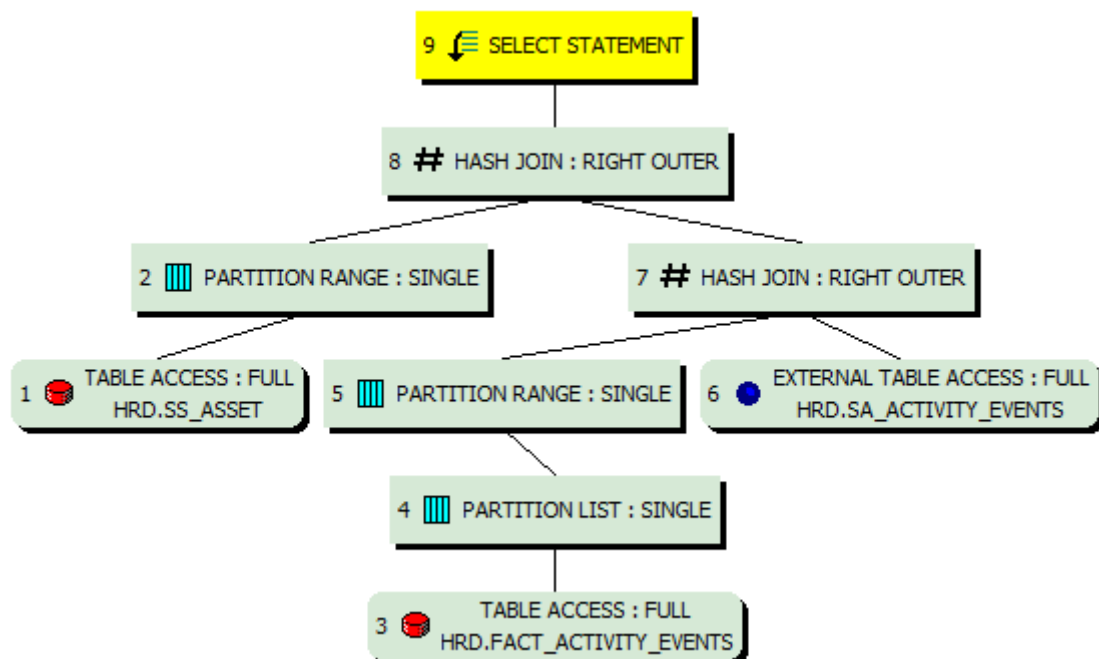


Figure 9.2. *Program Watched* transformation execution plan

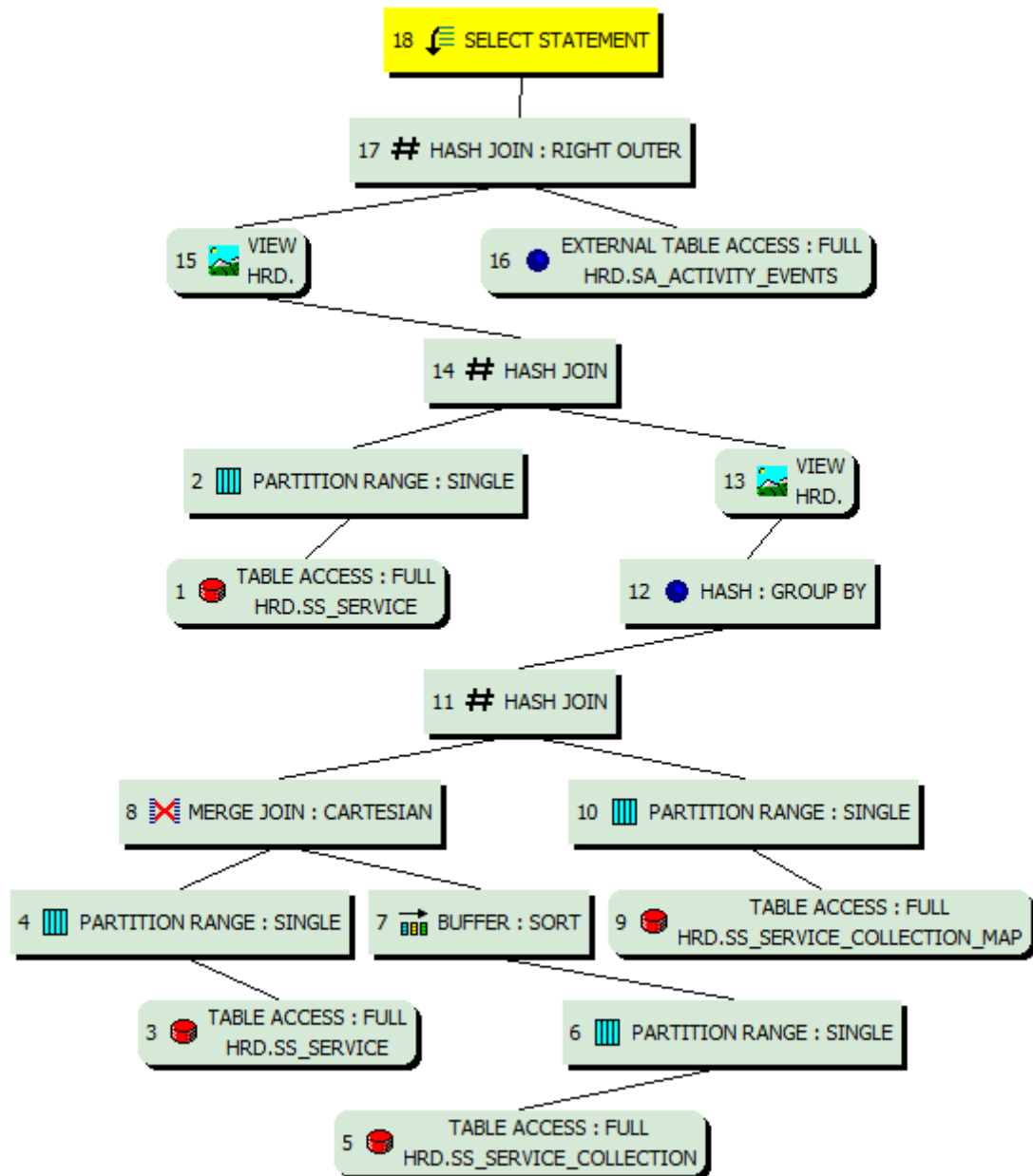


Figure 9.3. *DVR Events* transformation execution plan

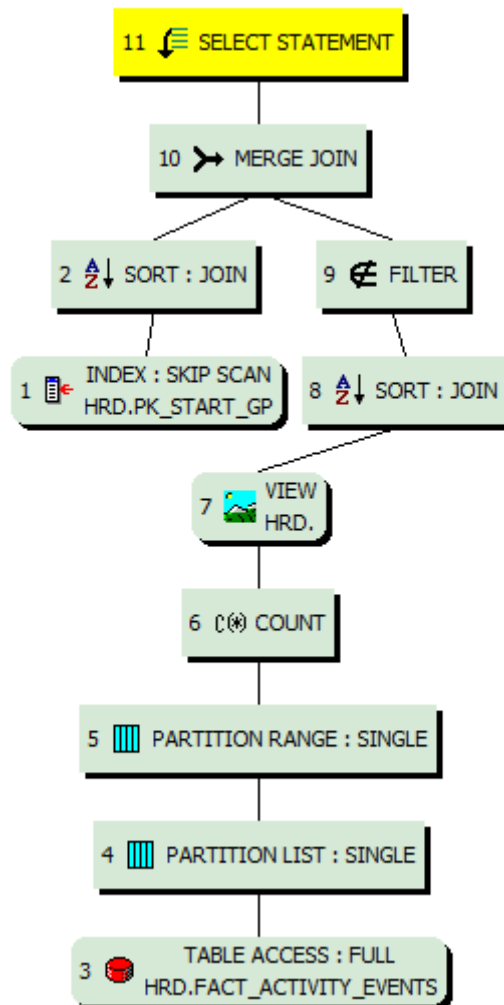


Figure 9.4. *Event Segmentation* execution plan

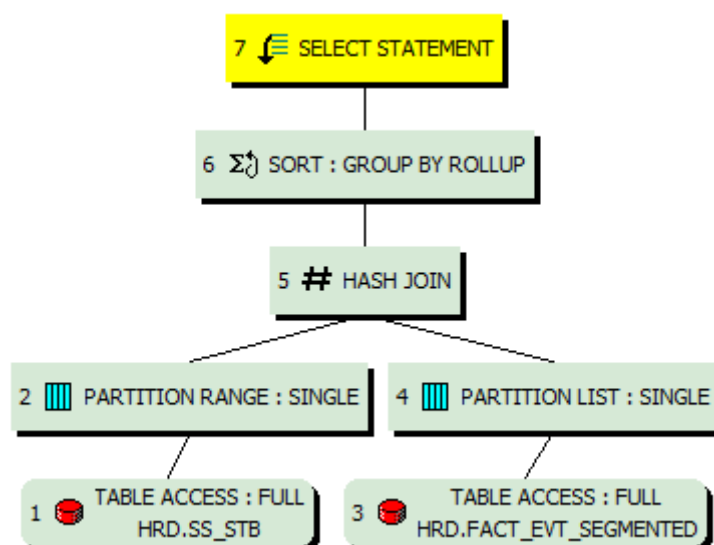


Figure 9.5. *Audiences Aggregation* execution plan

Appendix E. HADOOP CLUSTER IMPLEMENTATION

Appendix E.1. STORAGE OPTIONS ANALYSIS

In order to achieve a better balance between performance and storage size, in the implementation of the physical model, we need to consider and test several storage options available in Hive. As it was done for Oracle, in Appendix D.1, here we performed the same tests and analyzed the results.

The storage options available for creating tables in Hive are vast and we analyzed some of the most frequently used. Besides the storage of data as simple text files, we took a look at the Record Columnar File (RCFile) and at the Optimized Row Columnar (ORC), the latter with two possible compression formats, Zlib (ORC-Z) and Snappy (ORC-S). The ORC format, by design, already compresses data by grouping rows of data in stripes, stores metadata and it is optimized to handle projection clauses defined on Hive statements thus enabling efficient reads that require just a subset of columns (Huai et al., 2014).

The first test comprises of loading and transforming a compressed text file of 32MB (its uncompressed size is 200MB) containing *Channel Tune* events (section 4.5.6.1). Because we are loading a single compressed file all the executions used only a single Mapper.

File format	Execution time (s)	Size (MB)	Compression ratio
Text	44	244	1.0 : 1.0
RCFile	43	220	1.1: 1.0
ORC	46	40	6.1: 1.0
ORC-Z	48	17	14.1 : 1.0
ORC-S	49	26	9.5 : 1.0

Table 9.46. Hive 'write' compression test

From the analysis of Table 9.46 we can see that the ORC format, even without any extra compression, offers great compression ratios. Ratios that can even be largely improved by using, for example, the Zlib compression. Regarding performance, and due to the simplicity of the test, our findings are not conclusive, even though a small degradation can be observed in more compressed formats, when compared to less compressed formats.

Using the tables generated by our first test as source, the second test attempts to assess the performance implications in a process that, not only writes its output in a compressed format, but also reads its source data from compressed formats. The process used as test subject corresponds to the *Event Segmentation* described in section 4.5.6.3.

Compression	Execution time (s)					Size (MB)	
From/To	Text	RCFile	ORC	ORC-Z	ORC-S	Source	Target
Text	77	59	58	59	55	244	831
RCFile	63	60	55	54	53	220	775
ORC	74	73	69	61	60	40	81
ORC-Z	174	161	161	161	149	17	23
ORC-S	170	168	162	163	166	26	33

Table 9.47. Hive 'read/write' compression test

Table 9.47 shows the results of the *Event Segmentation* process in the possible combinations of read and write using the several file formats subject of the analysis. Since this test only used one file as source, the performance of uncompressed data against compressed data cannot be performed in a linear way. With just one file, compressed formats (ORC-Z and ORC-S) cannot be split for processing between several nodes and therefore the timings in Table 9.47 were obtained with ten Mapper tasks for the first three formats (Text, RCFile and ORC) while only one Mapper task was used for the compressed ORC formats (Zlib and Snappy). Regarding processing parallelization, it is also important to highlight that, while for text files the split is done by chunks of data, defined by their size, ORC files are split by stripes.

From the analysis of Table 9.47 it is obvious the enormous gains we can obtain by using the ORC format. Gains that can even be extended by using the extra compression of Zlib or Snappy. Zlib once again is able to compress the original data to smaller sizes and it is slightly faster than Snappy for reads but slower for writes.

With the gathered results, we decided to store our fact tables using ORC without any extra compression. When compared to the best compression obtained in the tests performed for Oracle (Appendix D.1) the ORC format will already give us enormous savings in storage space.

Appendix E.2. STAGING AREA TABLES

SA_ACTIVITY_EVENTS

```
CREATE EXTERNAL TABLE IF NOT EXISTS SA_ACTIVITY_EVENTS
(
  COD_DATE          INT,
  SOURCE_TIMESTAMP  TIMESTAMP,
  STB_ID            VARCHAR(100),
  STB_TYPE          VARCHAR(100),
  EVENT_TYPE        SMALLINT,
  CHANNEL_ID        VARCHAR(100),
  CHANNEL_NBR       INT,
  CONTENT_ID        VARCHAR(100),
  STATION_ID        VARCHAR(100),
  VIEW_MODE         VARCHAR(100),
  DURATION          SMALLINT,
  EXPIRATION_DATE   TIMESTAMP,
  ACTION            VARCHAR(100),
  ACTION_TIMESTAMP  TIMESTAMP,
  ACTION_STATE      VARCHAR(100),
  CATEGORY          VARCHAR(100),
  APP_NAME          VARCHAR(100),
  MENU_ID           VARCHAR(100),
  RESOLUTION        VARCHAR(100),
  CULTURE           VARCHAR(100),
  DYNAMIC           VARCHAR(100),
  RECURRING         VARCHAR(100),
  INSTANCE_OF_RECURRING VARCHAR(100),
  FREQUENCY         VARCHAR(100),
  MANUAL_DELETION   VARCHAR(100),
  BYTES            SMALLINT,
  TUNE_ID           VARCHAR(100)
)
COMMENT 'Activity Logs Source File'
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
  "separatorChar" = "\;",
  "quoteChar"     = "\"",
  "escapeChar"    = "\\"
)
STORED AS TEXTFILE
LOCATION '/user/hrd/sa/sa_activity_events';
```

SA_ASSET

```
CREATE EXTERNAL TABLE IF NOT EXISTS SA_ASSET
(
  COD_DATE          INT,
  SOURCE            VARCHAR(100),
  ASSET_ID          VARCHAR(100),
  TITLE            VARCHAR(500),
  DESCRIPTION       VARCHAR(1024),
  TYPE             VARCHAR(25),
  LANGUAGE          VARCHAR(20),
  COUNTRY_REGION   VARCHAR(20),
  PROVIDER_NAME     VARCHAR(100),
  GENRE            VARCHAR(100),
  SERVICE_COLLECTION_ID VARCHAR(50),
  DURATION          SMALLINT,
  RELEASE_YEAR      SMALLINT,
  STUDIO           VARCHAR(100),
  PRICE            DECIMAL(5,2),
  RATING           VARCHAR(100)
)
```

```

COMMENT 'Asset list'
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
  "separatorChar" = ";",
  "quoteChar"     = "\"",
  "escapeChar"    = "\\"
)
STORED AS TEXTFILE
LOCATION '/user/hrd/sa/sa_asset';

```

SA_CHANNEL_MAP

```

CREATE EXTERNAL TABLE IF NOT EXISTS SA_CHANNEL_MAP
(
  COD_DATE      INT,
  SOURCE        VARCHAR(100),
  CHANNEL_MAP_ID VARCHAR(100),
  DESCRITPION   VARCHAR(100),
  FLG_DEFAULT   TINYINT
)
COMMENT 'Channel Maps'
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
  "separatorChar" = ";",
  "quoteChar"     = "\"",
  "escapeChar"    = "\\"
)
STORED AS TEXTFILE
LOCATION '/user/hrd/sa/sa_channel_map';

```

SA_GROUP

```

CREATE EXTERNAL TABLE IF NOT EXISTS SA_GROUP
(
  COD_DATE      INT,
  SOURCE        VARCHAR(100),
  GROUP_ID      VARCHAR(100),
  CHANNEL_MAP_ID VARCHAR(100),
  INTERNAL_ID   VARCHAR(100)
)
COMMENT 'Groups'
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
  "separatorChar" = ";",
  "quoteChar"     = "\"",
  "escapeChar"    = "\\"
)
STORED AS TEXTFILE
LOCATION '/user/hrd/sa/sa_group';

```

SA_PROGRAM

```

CREATE EXTERNAL TABLE IF NOT EXISTS SA_PROGRAM
(
  COD_DATE      INT,
  SOURCE        VARCHAR(100),
  PROGRAM_ID    VARCHAR(100),
  DESCRIPTION    VARCHAR(300),
  SERVICE_ID    VARCHAR(100)
)
COMMENT 'Program list'
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (

```

```

    "separatorChar" = ";",
    "quoteChar"     = "\"",
    "escapeChar"    = "\\"
)
STORED AS TEXTFILE
LOCATION '/user/hrd/sa/sa_program';

```

SA_SERVICE

```

CREATE EXTERNAL TABLE IF NOT EXISTS SA_SERVICE
(
    COD_DATE          INT,
    SOURCE            VARCHAR(100),
    SERVICE_ID        VARCHAR(100),
    DESCRIPTION       VARCHAR(100),
    VIEW_MODE        VARCHAR(100),
    INTENT            VARCHAR(100),
    TYPE              VARCHAR(100),
    MULTICAST_GRP_IP_ADDR VARCHAR(100),
    VIDEO_BITRATE     VARCHAR(100),
    AUDIO_BITRATE     VARCHAR(100),
    AUDIO_CODEC       VARCHAR(100),
    PROCESS_ID        VARCHAR(100),
    PROCESS_ID_CODE   VARCHAR(100)
)
COMMENT 'Services'
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
    "separatorChar" = ";",
    "quoteChar"     = "\"",
    "escapeChar"    = "\\"
)
STORED AS TEXTFILE
LOCATION '/user/hrd/sa/sa_service';

```

SA_SERVICE_COLLECTION

```

CREATE EXTERNAL TABLE IF NOT EXISTS SA_SERVICE_COLLECTION
(
    COD_DATE          INT,
    SOURCE            VARCHAR(100),
    ID                VARCHAR(100),
    SERVICE_COLLECTION_ID VARCHAR(100),
    DESCRIPTION       VARCHAR(100),
    EPG_ID            VARCHAR(100)
)
COMMENT 'Service Collections'
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
    "separatorChar" = ";",
    "quoteChar"     = "\"",
    "escapeChar"    = "\\"
)
STORED AS TEXTFILE
LOCATION '/user/hrd/sa/sa_service_collection';

```

SA_SERVICE_COLLECTION_MAP

```

CREATE EXTERNAL TABLE IF NOT EXISTS SA_SERVICE_COLLECTION_MAP
(
    COD_DATE          INT,
    SOURCE            VARCHAR(100),
    SERVICE_ID        VARCHAR(100),

```

```

SERVICE_COLLECTION_ID VARCHAR(100),
TYPE                   VARCHAR(100),
SERVICE_ORDER        SMALLINT
)
COMMENT 'Service Collection Mappings'
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
  "separatorChar" = ";",
  "quoteChar"     = "\"",
  "escapeChar"    = "\\"
)
STORED AS TEXTFILE
LOCATION '/user/hrd/sa/sa_service_collection_map';

```

SA_STB

```

CREATE EXTERNAL TABLE IF NOT EXISTS SA_STB
(
  COD_DATE      INT,
  SOURCE        VARCHAR(100),
  CLIENT_ID     VARCHAR(100),
  EXTERNAL_ID   VARCHAR(100),
  SUBSCRIBER_ID VARCHAR(100),
  STATUS        VARCHAR(100),
  VERSION       VARCHAR(100)
)
COMMENT 'STBs inventory'
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
  "separatorChar" = ";",
  "quoteChar"     = "\"",
  "escapeChar"    = "\\"
)
STORED AS TEXTFILE
LOCATION '/user/hrd/sa/sa_stb';

```

SA_SUBSCRIBER_GROUP_MAP

```

CREATE EXTERNAL TABLE IF NOT EXISTS SA_SUBSCRIBER_GROUP_MAP
(
  COD_DATE      INT,
  SOURCE        VARCHAR(100),
  SUBSCRIBER_ID VARCHAR(100),
  GROUP_ID      VARCHAR(100)
)
COMMENT 'Subscriber-Group Mapping'
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
  "separatorChar" = ";",
  "quoteChar"     = "\"",
  "escapeChar"    = "\\"
)
STORED AS TEXTFILE
LOCATION '/user/hrd/sa/sa_subscriber_group_map';

```

SA_TV_CHANNEL

```

CREATE EXTERNAL TABLE IF NOT EXISTS SA_TV_CHANNEL
(
  COD_DATE      INT,
  SOURCE        VARCHAR(100),
  TUNER_POSITION INT,
  SERVICE_COLLECTION_ID VARCHAR(100),

```



```

CHANNEL_MAP_ID          VARCHAR(100)
)
COMMENT 'TV Channels'
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
  "separatorChar" = ";",
  "quoteChar"     = "\"",
  "escapeChar"    = "\\"
)
STORED AS TEXTFILE
LOCATION '/user/hrd/sa/sa_tv_channel';

```

Appendix E.3. SUPPORT TABLES

```

LU_DATE
CREATE TABLE IF NOT EXISTS LU_DATE
(
  COD_DATE      INT,
  DSC_DATE      VARCHAR(10),
  COD_WEEK      INT,
  COD_MONTH     INT,
  COD_QUARTER   INT,
  COD_SEMESTER  INT,
  COD_YEAR      INT
)
COMMENT 'Dates'
STORED AS ORC
LOCATION '/user/hrd/dw/lu_date'
TBLPROPERTIES ('orc.compress'='NONE');

```

```

LU_START_GP
CREATE TABLE IF NOT EXISTS LU_START_GP
(
  COD_START_GP   INT,
  COD_GP_DURATION SMALLINT,
  DSC_START_GP   VARCHAR(20)
)
COMMENT 'Granularity Periods'
STORED AS ORC
LOCATION '/user/hrd/dw/lu_start_gp'
TBLPROPERTIES ('orc.compress'='NONE');

```

Appendix E.4. INVENTORY TABLES

```

SS_ASSET
CREATE TABLE IF NOT EXISTS SS_ASSET
(
  ASSET_ID          VARCHAR(100),
  TITLE             VARCHAR(500),
  DESCRIPTION        VARCHAR(1024),
  TYPE              VARCHAR(25),
  LANGUAGE           VARCHAR(20),
  COUNTRY_REGION    VARCHAR(20),
  PROVIDER_NAME      VARCHAR(100),
  GENRE              VARCHAR(100),
  SERVICE_COLLECTION_ID VARCHAR(50),
  DURATION           SMALLINT,
  RELEASE_YEAR       SMALLINT,
  STUDIO            VARCHAR(100),
  PRICE             DECIMAL(5,2),
  RATING            VARCHAR(100)
)

```

```
)
COMMENT 'Asset list'
PARTITIONED BY (COD_DATE INT)
STORED AS ORC
LOCATION '/user/hrd/dw/ss_asset'
TBLPROPERTIES ('orc.compress'='NONE');
```

Data loading

```
INSERT OVERWRITE TABLE SS_ASSET PARTITION (COD_DATE = ${hiveconf:COD_DATE})
SELECT CASE WHEN ASSET_ID = '' THEN NULL ELSE ASSET_ID END,
       CASE WHEN TITLE = '' THEN NULL ELSE TITLE END,
       CASE WHEN DESCRIPTION = '' THEN NULL ELSE DESCRIPTION END,
       CASE WHEN TYPE = '' THEN NULL ELSE TYPE END,
       CASE WHEN LANGUAGE = '' THEN NULL ELSE LANGUAGE END,
       CASE WHEN COUNTRY_REGION = '' THEN NULL ELSE COUNTRY_REGION END,
       CASE WHEN PROVIDER_NAME = '' THEN NULL ELSE PROVIDER_NAME END,
       CASE WHEN GENRE = '' THEN NULL ELSE GENRE END,
       CASE WHEN SERVICE_COLLECTION_ID = '' THEN NULL ELSE SERVICE_COLLECTION_ID END,
       CASE WHEN DURATION = '' THEN 0 ELSE DURATION END,
       CASE WHEN RELEASE_YEAR = '' THEN NULL ELSE RELEASE_YEAR END,
       CASE WHEN STUDIO = '' THEN NULL ELSE STUDIO END,
       CASE WHEN PRICE = '' THEN 0 ELSE PRICE END,
       CASE WHEN RATING = '' THEN NULL ELSE RATING END
FROM SA_ASSET
WHERE COD_DATE = ${hiveconf:COD_DATE};
```

SS_CHANNEL_MAP

```
CREATE TABLE SS_CHANNEL_MAP
(
  CHANNEL_MAP_ID VARCHAR(100),
  DESCRIPTION VARCHAR(100),
  FLG_DEFAULT TINYINT
)
PARTITIONED BY (COD_DATE INT)
STORED AS ORC
LOCATION '/user/hrd/dw/ss_channel_map'
TBLPROPERTIES ('orc.compress'='NONE');
```

Data loading

```
INSERT OVERWRITE TABLE SS_CHANNEL_MAP PARTITION (COD_DATE = ${hiveconf:COD_DATE})
SELECT CHANNEL_MAP_ID,
       DESCRIPTION,
       FLG_DEFAULT
FROM SA_CHANNEL_MAP
WHERE COD_DATE = ${hiveconf:COD_DATE};
```

SS_GROUP

```
CREATE TABLE IF NOT EXISTS SS_GROUP
(
  GROUP_ID VARCHAR(100),
  CHANNEL_MAP_ID VARCHAR(100),
  INTERNAL_ID VARCHAR(100)
)
PARTITIONED BY (COD_DATE INT)
STORED AS ORC
LOCATION '/user/hrd/dw/ss_group'
TBLPROPERTIES ('orc.compress'='NONE');
```

Data loading

```
INSERT OVERWRITE TABLE SS_GROUP PARTITION (COD_DATE = ${hiveconf:COD_DATE})
SELECT GROUP_ID,
       CASE WHEN CHANNEL_MAP_ID = '' THEN NULL ELSE CHANNEL_MAP_ID END,
       CASE WHEN INTERNAL_ID = '' THEN NULL ELSE INTERNAL_ID END
```

```
FROM SA_GROUP
WHERE COD_DATE = ${hiveconf:COD_DATE};
```

SS_PROGRAM

```
CREATE TABLE IF NOT EXISTS SS_PROGRAM
(
  PROGRAM_ID VARCHAR(100),
  DESCRIPTION VARCHAR(300),
  SERVICE_ID VARCHAR(100)
)
COMMENT 'Program list'
PARTITIONED BY (COD_DATE INT)
STORED AS ORC
LOCATION '/user/hrd/dw/ss_program'
TBLPROPERTIES ('orc.compress'='NONE');
```

Data loading

```
INSERT OVERWRITE TABLE SS_PROGRAM PARTITION (COD_DATE = ${hiveconf:COD_DATE})
SELECT CASE WHEN PROGRAM_ID = '' THEN NULL ELSE PROGRAM_ID END,
       CASE WHEN DESCRIPTION = '' THEN NULL ELSE DESCRIPTION END,
       CASE WHEN SERVICE_ID = '' THEN NULL ELSE SERVICE_ID END
FROM SA_PROGRAM
WHERE COD_DATE = ${hiveconf:COD_DATE};

-- Manually insert the 'Program Measures Totalizer' that is used to aggregate global
values from all the Programs
INSERT INTO TABLE SS_PROGRAM
PARTITION (COD_DATE = ${hiveconf:COD_DATE})
(program_id, description, service_id)
VALUES
('0', 'Program Measures Totalizer', 'GlobalIPTVTVService');
```

SS_SERVICE

```
CREATE TABLE SS_SERVICE
(
  SERVICE_ID VARCHAR(100),
  DESCRIPTION VARCHAR(100),
  VIEW_MODE VARCHAR(100),
  INTENT VARCHAR(100),
  TYPE VARCHAR(100),
  MULTICAST_GRP_IP_ADDR VARCHAR(100),
  VIDEO_BITRATE VARCHAR(100),
  AUDIO_BITRATE VARCHAR(100),
  AUDIO_CODEC VARCHAR(100),
  PROCESS_ID VARCHAR(100),
  PROCESS_ID_CODE VARCHAR(100)
)
PARTITIONED BY (COD_DATE INT)
STORED AS ORC
LOCATION '/user/hrd/dw/ss_service'
TBLPROPERTIES ('orc.compress'='NONE');
```

Data loading

```
INSERT OVERWRITE TABLE SS_SERVICE PARTITION (COD_DATE = ${hiveconf:COD_DATE})
SELECT CASE WHEN SERVICE_ID = '' THEN NULL ELSE SERVICE_ID END,
       CASE WHEN DESCRIPTION = '' THEN NULL ELSE
RTRIM(TRANSLATE(REGEXP_REPLACE(DESCRIPTION, '(pip)|(PIP)|(main)|(Main)', ''), '_ ', ' '))
END,
       CASE WHEN VIEW_MODE = '' THEN NULL ELSE VIEW_MODE END,
       CASE WHEN INTENT = '' THEN NULL ELSE INTENT END,
       CASE WHEN TYPE = '' THEN NULL ELSE TYPE END,
       CASE WHEN MULTICAST_GRP_IP_ADDR = '' THEN NULL ELSE MULTICAST_GRP_IP_ADDR END,
```

```

CASE WHEN VIDEO_BITRATE = '' THEN NULL ELSE VIDEO_BITRATE END,
CASE WHEN AUDIO_BITRATE = '' THEN NULL ELSE AUDIO_BITRATE END,
CASE WHEN AUDIO_CODEC = '' THEN NULL ELSE AUDIO_CODEC END,
CASE WHEN PROCESS_ID = '' THEN NULL ELSE PROCESS_ID END,
CASE WHEN PROCESS_ID_CODE = '' THEN NULL ELSE PROCESS_ID_CODE END
FROM SA_SERVICE
WHERE COD_DATE = ${hiveconf:COD_DATE};

-- Manually insert the Global IPTV TV Service that is used to aggregate global values
from all the TV Service channels
INSERT INTO TABLE SS_SERVICE
PARTITION (COD_DATE = ${hiveconf:COD_DATE})
(service_id, description)
VALUES
('GlobalIPTVTVService', 'Global TV Service');

```

SS_SERVICE_COLLECTION

```

CREATE TABLE SS_SERVICE_COLLECTION
(
  SERVICE_COLLECTION_ID VARCHAR(100),
  EPG_ID                 VARCHAR(100),
  DESCRIPTION            VARCHAR(100),
  ID                     VARCHAR(100)
)
PARTITIONED BY (COD_DATE INT)
STORED AS ORC
LOCATION '/user/hrd/dw/ss_service_collection'
TBLPROPERTIES ('orc.compress'='NONE');

```

Data loading

```

INSERT OVERWRITE TABLE SS_SERVICE_COLLECTION PARTITION (COD_DATE = ${hiveconf:COD_DATE})
SELECT CASE WHEN SERVICE_COLLECTION_ID = '' THEN NULL ELSE SERVICE_COLLECTION_ID END,
CASE WHEN EPG_ID = '' THEN NULL ELSE EPG_ID END,
CASE WHEN DESCRIPTION = '' THEN NULL ELSE DESCRIPTION END,
CASE WHEN ID = '' THEN NULL ELSE ID END
FROM SA_SERVICE_COLLECTION
WHERE COD_DATE = ${hiveconf:COD_DATE};

```

SS_SERVICE_COLLECTION_MAP

```

CREATE TABLE SS_SERVICE_COLLECTION_MAP
(
  SERVICE_COLLECTION_ID VARCHAR(100),
  SERVICE_ID             VARCHAR(100),
  TYPE                   VARCHAR(100),
  SERVICE_ORDER          SMALLINT
)
PARTITIONED BY (COD_DATE INT)
STORED AS ORC
LOCATION '/user/hrd/dw/ss_service_collection_map'
TBLPROPERTIES ('orc.compress'='NONE');

```

Data loading

```

INSERT OVERWRITE TABLE SS_SERVICE_COLLECTION_MAP PARTITION (COD_DATE =
${hiveconf:COD_DATE})
SELECT CASE WHEN SERVICE_COLLECTION_ID = '' THEN NULL ELSE SERVICE_COLLECTION_ID END,
CASE WHEN SERVICE_ID = '' THEN NULL ELSE SERVICE_ID END,
CASE WHEN TYPE = '' THEN NULL ELSE TYPE END,
SERVICE_ORDER
FROM SA_SERVICE_COLLECTION_MAP
WHERE COD_DATE = ${hiveconf:COD_DATE};

```

SS_STB

```
CREATE TABLE IF NOT EXISTS SS_STB
(
  STB_ID VARCHAR(100),
  EXTERNAL_ID VARCHAR(100),
  SUBSCRIBER_ID VARCHAR(100),
  STATUS VARCHAR(100),
  VERSION VARCHAR(100)
)
PARTITIONED BY (COD_DATE INT)
STORED AS ORC
LOCATION '/user/hrd/dw/ss_stb'
TBLPROPERTIES ('orc.compress'='NONE');
```

Data loading

```
INSERT OVERWRITE TABLE SS_STB PARTITION (COD_DATE = ${hiveconf:COD_DATE})
SELECT CASE WHEN CLIENT_ID = '' THEN NULL ELSE CLIENT_ID END STB_ID,
       CASE WHEN EXTERNAL_ID = '' THEN NULL ELSE EXTERNAL_ID END,
       CASE WHEN SUBSCRIBER_ID = '' THEN NULL ELSE SUBSCRIBER_ID END,
       CASE WHEN STATUS = '' THEN NULL ELSE STATUS END,
       CASE WHEN VERSION = '' THEN NULL ELSE VERSION END
FROM SA_STB
WHERE COD_DATE = ${hiveconf:COD_DATE};
```

SS_STB_GROUP_MAP

```
CREATE TABLE IF NOT EXISTS SS_STB_GROUP_MAP
(
  STB_ID VARCHAR(100),
  GROUP_ID VARCHAR(100)
)
PARTITIONED BY (COD_DATE INT)
STORED AS ORC
LOCATION '/user/hrd/dw/ss_stb_group_map'
TBLPROPERTIES ('orc.compress'='NONE');
```

Data loading

```
INSERT OVERWRITE TABLE SS_STB_GROUP_MAP PARTITION (COD_DATE = ${hiveconf:COD_DATE})
SELECT SUBSCRIBER_ID STB_ID,
       GROUP_ID
FROM SA_SUBSCRIBER_GROUP_MAP m
WHERE EXISTS (SELECT 1
              FROM SS_STB s
              WHERE m.SUBSCRIBER_ID = s.STB_ID
                  AND s.COD_DATE = ${hiveconf:COD_DATE})
AND m.COD_DATE = ${hiveconf:COD_DATE};
```

SS_SUBSCRIBER_GROUP_MAP

```
CREATE TABLE IF NOT EXISTS SS_SUBSCRIBER_GROUP_MAP
(
  SUBSCRIBER_ID VARCHAR(100),
  GROUP_ID VARCHAR(100)
)
PARTITIONED BY (COD_DATE INT)
STORED AS ORC
LOCATION '/user/hrd/dw/ss_subscriber_group_map'
TBLPROPERTIES ('orc.compress'='NONE');
```

Data loading

```
INSERT OVERWRITE TABLE SS_SUBSCRIBER_GROUP_MAP PARTITION (COD_DATE =
${hiveconf:COD_DATE})
SELECT m.SUBSCRIBER_ID,
       m.GROUP_ID
```

```

FROM SA_SUBSCRIBER_GROUP_MAP m,
     (SELECT DISTINCT SUBSCRIBER_ID
      FROM SS_STB
      WHERE COD_DATE = ${hiveconf:COD_DATE}) s
WHERE m.SUBSCRIBER_ID = s.SUBSCRIBER_ID
     AND m.COD_DATE = ${hiveconf:COD_DATE};

```

SS_TV_CHANNEL

```

CREATE TABLE SS_TV_CHANNEL
(
  TUNER_POSITION          INT,
  CHANNEL_MAP_ID          VARCHAR(100),
  SERVICE_COLLECTION_ID   VARCHAR(100)
)
PARTITIONED BY (COD_DATE INT)
STORED AS ORC
LOCATION '/user/hrd/dw/ss_tv_channel'
TBLPROPERTIES ('orc.compress'='NONE');

```

Data loading

```

INSERT OVERWRITE TABLE SS_TV_CHANNEL PARTITION (COD_DATE = ${hiveconf:COD_DATE})
SELECT TUNER_POSITION,
       CASE WHEN CHANNEL_MAP_ID = '' THEN NULL ELSE CHANNEL_MAP_ID END,
       CASE WHEN SERVICE_COLLECTION_ID= '' THEN NULL ELSE SERVICE_COLLECTION_ID END
FROM SA_TV_CHANNEL
WHERE COD_DATE = ${hiveconf:COD_DATE};

```

SS_MAP_CHANNEL_MAP_SERVICE

```

CREATE TABLE SS_MAP_CHANNEL_MAP_SERVICE
(
  CHANNEL_MAP_ID VARCHAR(100),
  TUNER_POSITION INT,
  SERVICE_ID VARCHAR(100),
  SERVICE_TYPE VARCHAR(100)
)
PARTITIONED BY (COD_DATE INT)
STORED AS ORC
LOCATION '/user/hrd/dw/ss_map_channel_map_service'
TBLPROPERTIES ('orc.compress'='NONE');

```

Data loading

```

INSERT OVERWRITE TABLE SS_MAP_CHANNEL_MAP_SERVICE PARTITION (COD_DATE =
${hiveconf:COD_DATE})
SELECT r.CHANNEL_MAP_ID,
       r.TUNER_POSITION,
       r.SERVICE_ID,
       r.SERVICE_TYPE
FROM (SELECT ch.CHANNEL_MAP_ID,
            ch.TUNER_POSITION,
            svc.SERVICE_ID,
            svc.TYPE SERVICE_TYPE,
            ROW_NUMBER()
              OVER (PARTITION BY ch.CHANNEL_MAP_ID, ch.TUNER_POSITION
                        ORDER BY CASE m.TYPE
                                WHEN 'FULLSCREEN_PRIMARY' THEN 1
                                WHEN 'FULLSCREEN_SECONDARY' THEN 2
                                END,
                        m.SERVICE_ORDER)
              ) RN
FROM SS_TV_CHANNEL ch,
     SS_SERVICE_COLLECTION col,

```



```

        ) c
        ON (stb.STB_ID = c.STB_ID)
        WHERE (c.CHANNEL_MAP_ID IS NOT NULL OR s.CHANNEL_MAP_ID IS NOT NULL)
        GROUP BY stb.STB_ID,
                COALESCE(c.CHANNEL_MAP_ID, s.CHANNEL_MAP_ID)
    ) m1
    GROUP BY STB_ID
    HAVING COUNT(*) = 1 -- To prevent Mediaroom configurations that have the same
STB in more than one Channel Map
    ) m2
    ON (uni.STB_ID = m2.STB_ID)
CROSS
JOIN (SELECT MAX(CHANNEL_MAP_ID) CHANNEL_MAP_ID
      FROM SS_CHANNEL_MAP
      WHERE FLG_DEFAULT = 1
      AND COD_DATE = ${hiveconf:COD_DATE}) d; -- Default Channel Map to STBs without
Channel Map or wrong configuration

```

Appendix E.5. FACT TABLES

FACT_ACTIVITY_EVENTS

```

CREATE TABLE FACT_ACTIVITY_EVENTS
(
    COD_DATE_GP          BIGINT,
    COD_START_GP         INT,
    SOURCE_TIMESTAMP     TIMESTAMP,
    STB_ID               VARCHAR(100),
    STB_TYPE             VARCHAR(100),
    SERVICE_TYPE         VARCHAR(100),
    CHANNEL_ID           VARCHAR(100),
    CHANNEL_NBR          INT,
    CONTENT_ID           VARCHAR(100),
    STATION_ID           VARCHAR(100),
    VIEW_MODE            VARCHAR(100),
    DURATION              SMALLINT,
    EXPIRATION_DATE      TIMESTAMP,
    ACTION               VARCHAR(100),
    ACTION_TIMESTAMP     TIMESTAMP,
    ACTION_STATE         VARCHAR(100),
    CATEGORY              VARCHAR(100),
    APP_NAME             VARCHAR(100),
    MENU_ID              VARCHAR(100),
    RESOLUTION           VARCHAR(100),
    CULTURE               VARCHAR(100),
    DYNAMIC               VARCHAR(100),
    RECURRING            VARCHAR(100),
    INSTANCE_OF_RECURRING VARCHAR(100),
    FREQUENCY            VARCHAR(100),
    MANUAL_DELETION      VARCHAR(100),
    BYTES                SMALLINT,
    TUNE_ID              VARCHAR(100)
)
PARTITIONED BY (COD_DATE INT, EVENT_TYPE SMALLINT)
STORED AS ORC
LOCATION '/user/hrd/dw/fact_activity_events'
TBLPROPERTIES ('orc.compress'='NONE');

```

Data loading – Event 100

```

ALTER TABLE FACT_ACTIVITY_EVENTS DROP IF EXISTS PARTITION (COD_DATE =
${hiveconf:COD_DATE}, EVENT_TYPE = 100) PURGE;

-- Channel Tune Event
INSERT INTO FACT_ACTIVITY_EVENTS
PARTITION (COD_DATE = ${hiveconf:COD_DATE}, EVENT_TYPE = 100)

```



```

SELECT CAST(FROM_UNIXTIME((UNIX_TIMESTAMP(SOURCE_TIMESTAMP, 'yyyy-MM-dd HH:mm:ss') -
(UNIX_TIMESTAMP(SOURCE_TIMESTAMP, 'yyyy-MM-dd HH:mm:ss') % 300)), 'yyyyMMddHHmm') AS
BIGINT) COD_DATE_GP,
    CAST(FROM_UNIXTIME((UNIX_TIMESTAMP(SOURCE_TIMESTAMP, 'yyyy-MM-dd HH:mm:ss') -
(UNIX_TIMESTAMP(SOURCE_TIMESTAMP, 'yyyy-MM-dd HH:mm:ss') % 300)), 'HHmm') AS INT)
COD_START_GP,
    sa.SOURCE_TIMESTAMP,
    ----- STB ID -----
    sa.STB_ID,
    ----- STB TYPE -----
    sa.STB_TYPE,
    ----- SERVICE TYPE -----
    srv.SERVICE_TYPE,
    ----- CHANNEL ID -----
    srv.SERVICE_ID,
    ----- CHANNEL NBR -----
    sa.CHANNEL_NBR,
    ----- CONTENT ID -----
    NULL,
    ----- STATION ID -----
    sa.STATION_ID,
    ----- VIEW MODE -----
    sa.VIEW_MODE,
    ----- DURATION -----
    sa.DURATION,
    ----- EXPIRATION_DATE -----
    NULL,
    ----- ACTION -----
    'Channel Tune',
    ----- ACTION TIMESTAMP -----
    NULL,
    ----- ACTION STATE -----
    sa.ACTION_STATE,
    ----- CATEGORY -----
    NULL,
    ----- APP NAME -----
    NULL,
    ----- MENU ID -----
    NULL,
    ----- RESOLUTION -----
    NULL,
    ----- CULTURE -----
    NULL,
    ----- DYNAMIC -----
    NULL,
    ----- RECURRING -----
    NULL,
    ----- INSTANCE OF RECURRING -----
    NULL,
    ----- FREQUENCY -----
    NULL,
    ----- MANUAL DELETION -----
    NULL,
    ----- BYTES -----
    NULL,
    ----- TUNE ID -----
    sa.TUNE_ID
FROM (SELECT evt.*,
    chanmap.CHANNEL_MAP_ID
    FROM SA_ACTIVITY_EVENTS evt
    LEFT OUTER
    JOIN SS_MAP_STB_CHANNEL_MAP chanmap
    ON (evt.STB_ID = chanmap.STB_ID AND
        chanmap.COD_DATE = CAST(${hiveconf:COD_DATE_INV} AS INT)
    )
    WHERE evt.EVENT_TYPE = 100

```

```

        AND evt.COD_DATE = CAST(${hiveconf:COD_DATE} AS INT)
    ) sa
LEFT OUTER
JOIN SS_MAP_CHANNEL_MAP_SERVICE srv
ON (sa.CHANNEL_NBR = srv.TUNER_POSITION AND
    sa.CHANNEL_MAP_ID = srv.CHANNEL_MAP_ID AND
    srv.COD_DATE = CAST(${hiveconf:COD_DATE_INV} AS INT)
);

```

Data loading – Event 101

```

ALTER TABLE FACT_ACTIVITY_EVENTS DROP IF EXISTS PARTITION (COD_DATE =
${hiveconf:COD_DATE}, EVENT_TYPE = 101) PURGE;

```

-- Set-top Box Power (On/Off) Event

```

INSERT INTO FACT_ACTIVITY_EVENTS
PARTITION (COD_DATE = ${hiveconf:COD_DATE}, EVENT_TYPE = 101)
SELECT
    CAST(FROM_UNIXTIME((UNIX_TIMESTAMP(sa.SOURCE_TIMESTAMP, 'yyyy-MM-dd HH:mm:ss') -
(UNIX_TIMESTAMP(sa.SOURCE_TIMESTAMP, 'yyyy-MM-dd HH:mm:ss') % 300)), 'yyyyMMddHHmm') AS
BIGINT) COD_DATE_GP,
    CAST(FROM_UNIXTIME((UNIX_TIMESTAMP(sa.SOURCE_TIMESTAMP, 'yyyy-MM-dd HH:mm:ss') -
(UNIX_TIMESTAMP(sa.SOURCE_TIMESTAMP, 'yyyy-MM-dd HH:mm:ss') % 300)), 'HHmm') AS INT)
COD_START_GP,
    sa.SOURCE_TIMESTAMP,
    sa.STB_ID,
    sa.STB_TYPE,
    ----- SERVICE TYPE -----
    NULL,
    ----- CHANNEL ID -----
    NULL,
    ----- CHANNEL NBR -----
    NULL,
    ----- CONTENT ID -----
    NULL,
    ----- STATION ID -----
    NULL,
    ----- VIEW MODE -----
    NULL,
    ----- DURATION -----
    NULL,
    ----- EXPIRATION_DATE -----
    NULL,
    ----- ACTION -----
    sa.ACTION,
    ----- ACTION_TIMESTAMP -----
    sa.SOURCE_TIMESTAMP,
    ----- ACTION_STATE -----
    1,
    ----- CATEGORY -----
    NULL,
    ----- APP_NAME -----
    NULL,
    ----- MENU_ID -----
    NULL,
    ----- RESOLUTION -----
    NULL,
    ----- CULTURE -----
    NULL,
    ----- DYNAMIC -----
    NULL,
    ----- RECURRING -----
    NULL,
    ----- INSTANCE OF RECURRING -----
    NULL,
    ----- FREQUENCY -----
    NULL,

```

```

----- MANUAL DELETION -----
NULL,
----- BYTES -----
NULL,
----- TUNE ID -----
NULL
FROM SA_ACTIVITY_EVENTS sa
WHERE sa.EVENT_TYPE = 101
AND sa.COD_DATE = CAST(${hiveconf:COD_DATE} AS INT);

```

Data loading – Event 104

```

ALTER TABLE FACT_ACTIVITY_EVENTS DROP IF EXISTS PARTITION (COD_DATE =
${hiveconf:COD_DATE}, EVENT_TYPE = 104) PURGE;

-- Event 104: Trick State
INSERT INTO FACT_ACTIVITY_EVENTS
PARTITION (COD_DATE = ${hiveconf:COD_DATE}, EVENT_TYPE = 104)
(
cod_date_gp,
cod_start_gp,
source_timestamp,
stb_id,
stb_type,
service_type,
content_id,
action,
action_timestamp,
action_state
)
SELECT
CAST(FROM_UNIXTIME((UNIX_TIMESTAMP(sa.SOURCE_TIMESTAMP, 'yyyy-MM-dd HH:mm:ss') -
(UNIX_TIMESTAMP(sa.SOURCE_TIMESTAMP, 'yyyy-MM-dd HH:mm:ss') % 300)), 'yyyyMMddHHmm') AS
BIGINT) COD_DATE_GP,
CAST(FROM_UNIXTIME((UNIX_TIMESTAMP(sa.SOURCE_TIMESTAMP, 'yyyy-MM-dd HH:mm:ss') -
(UNIX_TIMESTAMP(sa.SOURCE_TIMESTAMP, 'yyyy-MM-dd HH:mm:ss') % 300)), 'HHmm') AS INT)
COD_START_GP,
sa.SOURCE_TIMESTAMP,
sa.STB_ID,
sa.STB_TYPE,
CASE WHEN vod.ASSET_ID IS NOT NULL
THEN 'VOD'
ELSE typ.SERVICE_TYPE
END SERVICE_TYPE,
sa.CONTENT_ID,
sa.ACTION,
sa.SOURCE_TIMESTAMP ACTION_TIMESTAMP,
1 ACTION_STATE
FROM SA_ACTIVITY_EVENTS sa
LEFT OUTER
JOIN SS_ASSET vod
ON (sa.CONTENT_ID = vod.ASSET_ID AND
vod.COD_DATE = CAST(${hiveconf:COD_DATE_INV} AS INT))
LEFT OUTER
JOIN (SELECT col.ID,
MAX(svc.TYPE) SERVICE_TYPE
FROM SS_SERVICE_COLLECTION col,
SS_SERVICE_COLLECTION_MAP m,
SS_SERVICE svc
WHERE col.COD_DATE = CAST(${hiveconf:COD_DATE_INV} AS INT)
AND col.SERVICE_COLLECTION_ID = m.SERVICE_COLLECTION_ID
AND m.COD_DATE = CAST(${hiveconf:COD_DATE_INV} AS INT)
AND m.SERVICE_ID = svc.SERVICE_ID
AND svc.COD_DATE = CAST(${hiveconf:COD_DATE_INV} AS INT)
AND svc.VIEW_MODE NOT LIKE 'PIP%'
GROUP BY col.ID
) typ

```

```

ON (sa.CONTENT_ID = typ.ID)
WHERE sa.EVENT_TYPE = 104
AND sa.COD_DATE = CAST(${hiveconf:COD_DATE} AS INT);

```

Data loading – Event 114

```

ALTER TABLE FACT_ACTIVITY_EVENTS DROP IF EXISTS PARTITION (COD_DATE =
${hiveconf:COD_DATE}, EVENT_TYPE = 114) PURGE;

-- Program Transition Event
INSERT INTO FACT_ACTIVITY_EVENTS
PARTITION (COD_DATE = ${hiveconf:COD_DATE}, EVENT_TYPE = 114)
SELECT
    CAST(FROM_UNIXTIME((UNIX_TIMESTAMP(sa.SOURCE_TIMESTAMP, 'yyyy-MM-dd HH:mm:ss') -
(UNIX_TIMESTAMP(sa.SOURCE_TIMESTAMP, 'yyyy-MM-dd HH:mm:ss') % 300)), 'yyyyMMddHHmm') AS
BIGINT) COD_DATE_GP,
    CAST(FROM_UNIXTIME((UNIX_TIMESTAMP(sa.SOURCE_TIMESTAMP, 'yyyy-MM-dd HH:mm:ss') -
(UNIX_TIMESTAMP(sa.SOURCE_TIMESTAMP, 'yyyy-MM-dd HH:mm:ss') % 300)), 'HHmm') AS INT)
COD_START_GP,
    sa.SOURCE_TIMESTAMP,
    sa.STB_ID,
    sa.STB_TYPE,
    ----- SERVICE TYPE -----
COALESCE(fact.SERVICE_TYPE, vod.TYPE),
    ----- CHANNEL ID -----
    -- CHANNEL_ID for EVENT_TYPE 114 is the SERVICE_ID of the matching EVENT_TYPE 100
fact.SERVICE_ID,
    ----- CHANNEL NBR -----
NULL,
    ----- CONTENT ID -----
sa.CONTENT_ID,
    ----- STATION ID -----
NULL,
    ----- VIEW MODE -----
COALESCE(sa.VIEW_MODE, fact.VIEW_MODE),
    ----- DURATION -----
sa.DURATION,
    ----- EXPIRATION_DATE -----
NULL,
    ----- ACTION -----
'Program Transition',
    ----- ACTION_TIMESTAMP -----
NULL, -- SOURCE_TIMESTAMP,
    ----- ACTION_STATE -----
1, -- Success
    ----- CATEGORY -----
NULL,
    ----- APP_NAME -----
NULL,
    ----- MENU_ID -----
NULL,
    ----- RESOLUTION -----
NULL,
    ----- CULTURE -----
NULL,
    ----- DYNAMIC -----
NULL,
    ----- RECURRING -----
NULL,
    ----- INSTANCE_OF_RECURRING -----
NULL,
    ----- FREQUENCY -----
NULL,
    ----- MANUAL_DELETION -----
NULL,
    ----- BYTES -----
NULL,

```

```

----- TUNE ID -----
sa.TUNE_ID
FROM SA_ACTIVITY_EVENTS sa
LEFT OUTER -- STB_ID/TUNE_ID is unique for each event 100
JOIN (SELECT STB_ID, TUNE_ID, CHANNEL_ID SERVICE_ID, SERVICE_TYPE, VIEW_MODE
      FROM FACT_ACTIVITY_EVENTS
      WHERE EVENT_TYPE = 100 -- Channel Tune
            AND COD_DATE BETWEEN CAST(${hiveconf:COD_DATE_PREV} AS INT) AND
CAST(${hiveconf:COD_DATE} AS INT)
      ) fact
ON (sa.STB_ID = fact.STB_ID AND sa.TUNE_ID = fact.TUNE_ID)
LEFT OUTER -- Joining with VoDs to get a better definition of the service type for VoDs
JOIN SS_ASSET vod
ON (vod.COD_DATE = CAST(${hiveconf:COD_DATE_INV} AS INT) AND
    sa.CONTENT_ID = vod.ASSET_ID)
WHERE sa.EVENT_TYPE = 114 -- Program Transition
      AND sa.COD_DATE = CAST(${hiveconf:COD_DATE} AS INT);

```

Data loading – Event DVR

```

-- Hive session parameters
SET hive.exec.dynamic.partition.mode=nonstrict;

ALTER TABLE FACT_ACTIVITY_EVENTS DROP IF EXISTS PARTITION (COD_DATE =
${hiveconf:COD_DATE}, EVENT_TYPE = 115) PURGE;
ALTER TABLE FACT_ACTIVITY_EVENTS DROP IF EXISTS PARTITION (COD_DATE =
${hiveconf:COD_DATE}, EVENT_TYPE = 116) PURGE;
ALTER TABLE FACT_ACTIVITY_EVENTS DROP IF EXISTS PARTITION (COD_DATE =
${hiveconf:COD_DATE}, EVENT_TYPE = 117) PURGE;
ALTER TABLE FACT_ACTIVITY_EVENTS DROP IF EXISTS PARTITION (COD_DATE =
${hiveconf:COD_DATE}, EVENT_TYPE = 118) PURGE;
ALTER TABLE FACT_ACTIVITY_EVENTS DROP IF EXISTS PARTITION (COD_DATE =
${hiveconf:COD_DATE}, EVENT_TYPE = 119) PURGE;
ALTER TABLE FACT_ACTIVITY_EVENTS DROP IF EXISTS PARTITION (COD_DATE =
${hiveconf:COD_DATE}, EVENT_TYPE = 120) PURGE;

-- DVR Events
INSERT INTO FACT_ACTIVITY_EVENTS
PARTITION (COD_DATE, EVENT_TYPE)
SELECT
    CAST(FROM_UNIXTIME((UNIX_TIMESTAMP(sa.SOURCE_TIMESTAMP, 'yyyy-MM-dd HH:mm:ss') -
(UNIX_TIMESTAMP(sa.SOURCE_TIMESTAMP, 'yyyy-MM-dd HH:mm:ss') % 300)), 'yyyyMMddHHmm') AS
BIGINT) COD_DATE_GP,
    CAST(FROM_UNIXTIME((UNIX_TIMESTAMP(sa.SOURCE_TIMESTAMP, 'yyyy-MM-dd HH:mm:ss') -
(UNIX_TIMESTAMP(sa.SOURCE_TIMESTAMP, 'yyyy-MM-dd HH:mm:ss') % 300)), 'HHmm') AS INT)
COD_START_GP,
    sa.SOURCE_TIMESTAMP,
    sa.STB_ID,
    sa.STB_TYPE,
    ----- SERVICE TYPE -----
    sm.SERVICE_TYPE,
    ----- CHANNEL ID -----
    sm.SERVICE_ID,
    ----- CHANNEL NBR -----
    NULL,
    ----- CONTENT ID -----
    sa.CONTENT_ID,
    ----- STATION ID -----
    sa.STATION_ID,
    ----- VIEW MODE -----
    NULL,
    ----- DURATION -----
CASE
    WHEN sa.EVENT_TYPE IN (115, 118, 120) THEN
        sa.DURATION
END DURATION,
    ----- EXPIRATION_DATE -----

```

```

NULL,
----- ACTION -----
CASE
  WHEN sa.EVENT_TYPE = 115 THEN 'DVR Start Recording'
  WHEN sa.EVENT_TYPE = 116 THEN 'DVR Abort Recording'
  WHEN sa.EVENT_TYPE = 117 THEN 'DVR Playback Recording'
  WHEN sa.EVENT_TYPE = 118 THEN 'DVR Schedule Recording'
  WHEN sa.EVENT_TYPE = 119 THEN 'DVR Delete Recording'
  WHEN sa.EVENT_TYPE = 120 THEN 'DVR Cancel Recording'
END ACTION,
----- ACTION_TIMESTAMP -----
CASE
  WHEN sa.EVENT_TYPE IN (116, 118, 119, 120) THEN
    sa.ACTION_TIMESTAMP
  ELSE
    sa.SOURCE_TIMESTAMP
END ACTION_TIMESTAMP,
----- ACTION_STATE -----
1,
----- CATEGORY -----
NULL,
----- APP_NAME -----
NULL,
----- MENU_ID -----
NULL,
----- RESOLUTION -----
NULL,
----- CULTURE -----
NULL,
----- DYNAMIC -----
CASE
  WHEN sa.EVENT_TYPE IN (115, 118, 120) THEN
    sa.DYNAMIC
END DYNAMIC,
----- RECURRING -----
CASE
  WHEN sa.EVENT_TYPE IN (115, 118, 120) THEN
    sa.RECURRING
END RECURRING,
----- INSTANCE_OF_RECURRING -----
CASE
  WHEN sa.EVENT_TYPE IN (120) THEN
    sa.INSTANCE_OF_RECURRING
END INSTANCE_OF_RECURRING,
----- FREQUENCY -----
CASE
  WHEN sa.EVENT_TYPE IN (118, 120) THEN
    sa.FREQUENCY
END FREQUENCY,
----- MANUAL_DELETION -----
CASE
  WHEN sa.EVENT_TYPE IN (119) THEN
    sa.MANUAL_DELETION
END MANUAL_DELETION,
----- BYTES -----
CASE
  WHEN sa.EVENT_TYPE IN (119) THEN
    sa.BYTES
END BYTES,
----- TUNE_ID -----
NULL,
sa.COD_DATE, -- Partition column
sa.EVENT_TYPE -- Partition column
FROM SA_ACTIVITY_EVENTS sa
LEFT OUTER
JOIN (SELECT svc.SERVICE_ID,

```

```

        svc.TYPE SERVICE_TYPE,
        e.EPG_ID
    FROM SS_SERVICE svc
    INNER JOIN (SELECT col.EPG_ID,
                      MAX(svc2.SERVICE_ID) SERVICE_ID
                FROM SS_SERVICE svc2,
                     SS_SERVICE_COLLECTION_MAP scm,
                     SS_SERVICE_COLLECTION col
                WHERE svc2.COD_DATE = CAST(${hiveconf:COD_DATE_INV} AS INT)
                      AND svc2.VIEW_MODE = 'FULLSCREEN'
                      AND svc2.SERVICE_ID = scm.SERVICE_ID
                      AND scm.COD_DATE = CAST(${hiveconf:COD_DATE_INV} AS INT)
                      AND scm.SERVICE_COLLECTION_ID = col.SERVICE_COLLECTION_ID
                      AND col.COD_DATE = CAST(${hiveconf:COD_DATE_INV} AS INT)
                GROUP BY col.EPG_ID) e
    ON (svc.SERVICE_ID = e.SERVICE_ID)
    WHERE svc.COD_DATE = CAST(${hiveconf:COD_DATE_INV} AS INT)
) sm
ON (sa.STATION_ID = sm.EPG_ID)
WHERE sa.EVENT_TYPE IN (115, 116, 117, 118, 119, 120) -- DVR related events
AND sa.COD_DATE = CAST(${hiveconf:COD_DATE} AS INT);

```

FACT_EVT_SEGMENTED

```

CREATE TABLE FACT_EVT_SEGMENTED
(
    COD_DATE_GP          BIGINT,
    COD_START_GP         INT,
    COD_GP_DURATION      SMALLINT,
    SERVICE_TYPE         VARCHAR(20),
    SERVICE_ID           VARCHAR(100),
    CONTENT_ID           VARCHAR(100),
    STB_ID               VARCHAR(100),
    SOURCE_TIMESTAMP     TIMESTAMP,
    DURATION             INT,
    TOTAL_DURATION       INT
)
PARTITIONED BY (COD_DATE INT, EVENT_TYPE SMALLINT)
STORED AS ORC
LOCATION '/user/hrd/dw/fact_evt_segmented'
TBLPROPERTIES ('orc.compress'='NONE');

```

Data loading

```

SET EVENT_TYPE = 100; -- The process can also be executed for event 114

ALTER TABLE FACT_EVT_SEGMENTED DROP IF EXISTS PARTITION (COD_DATE = ${hiveconf:COD_DATE},
EVENT_TYPE = ${hiveconf:EVENT_TYPE}) PURGE;

INSERT OVERWRITE TABLE FACT_EVT_SEGMENTED
PARTITION (COD_DATE = ${hiveconf:COD_DATE}, EVENT_TYPE = ${hiveconf:EVENT_TYPE})
SELECT *
FROM (SELECT
        gp.COD_DATE_GP,
        gp.COD_START_GP,
        gp.COD_GP_DURATION,
        evt.SERVICE_TYPE,
        evt.CHANNEL_ID SERVICE_ID,
        evt.CONTENT_ID,
        evt.STB_ID,
        evt.SOURCE_TIMESTAMP,
        CAST(LEAST(CAST(300 AS BIGINT), UNIX_TIMESTAMP(MY_END_DATE) -
UNIX_TIMESTAMP(gp.DATE_GP))
        -
        GREATEST(CAST(0 AS BIGINT), UNIX_TIMESTAMP(MY_START_DATE) -
UNIX_TIMESTAMP(gp.DATE_GP))

```

```

        AS INT) DURATION,
        evt.DURATION TOTAL_DURATION
    FROM (SELECT CAST(FROM_UNIXTIME((UNIX_TIMESTAMP(e1.MY_START_DATE, 'yyyy-MM-dd
HH:mm:ss') - (UNIX_TIMESTAMP(e1.MY_START_DATE, 'yyyy-MM-dd HH:mm:ss') % 300)),
'yyyyMMddHHmm') AS BIGINT) MY_START_DATE_GP,
        CAST(FROM_UNIXTIME((UNIX_TIMESTAMP(e1.MY_END_DATE, 'yyyy-MM-dd
HH:mm:ss') - (UNIX_TIMESTAMP(e1.MY_END_DATE, 'yyyy-MM-dd HH:mm:ss') % 300)),
'yyyyMMddHHmm') AS BIGINT) MY_END_DATE_GP,
        e1.*
    FROM (SELECT
        GREATEST(fi.SOURCE_TIMESTAMP,
CAST(CAST(FROM_UNIXTIME(UNIX_TIMESTAMP('${hiveconf:COD_DATE}', 'yyyyMMdd'), 'yyyy-MM-dd')
AS DATE) AS TIMESTAMP)) MY_START_DATE,

LEAST(CAST(FROM_UNIXTIME(UNIX_TIMESTAMP(fi.SOURCE_TIMESTAMP) + DURATION) AS TIMESTAMP),
CAST(CAST(
DATE_ADD(FROM_UNIXTIME(UNIX_TIMESTAMP('${hiveconf:COD_DATE}', 'yyyyMMdd'),
'yyyy-MM-dd HH:mm:ss'), 1) AS DATE) AS TIMESTAMP) ) MY_END_DATE,
        fi.*
    FROM FACT_ACTIVITY_EVENTS fi
    WHERE fi.COD_DATE BETWEEN ${hiveconf:COD_DATE_PREV} AND
${hiveconf:COD_DATE}
        AND fi.EVENT_TYPE IN (${hiveconf:EVENT_TYPE})
        AND fi.VIEW_MODE NOT LIKE 'PIP%'
    ) e1
    ) evt,
    (SELECT y.COD_DATE,
        CAST(y.COD_DATE AS BIGINT) * 10000 + y.COD_START_GP
    COD_DATE_GP,
        y.COD_START_GP,
        y.COD_GP_DURATION,
        FROM_UNIXTIME(UNIX_TIMESTAMP(CAST((y.COD_DATE * 10000 +
y.COD_START_GP) AS VARCHAR(20)), 'yyyyMMddHHmm'), 'yyyy-MM-dd HH:mm:ss') DATE_GP
    FROM (SELECT CAST(${hiveconf:COD_DATE_PREV} AS BIGINT) COD_DATE,
        COD_START_GP,
        COD_GP_DURATION,
        ROW_NUMBER() OVER (PARTITION BY NULL ORDER BY
    COD_START_GP) RN
        FROM LU_START_GP
        WHERE COD_GP_DURATION = 5
    ) y -- Previous day. We only need 3 hours of GPs because the
events are truncated to a maximum of 3 hours
    WHERE y.RN >= (1440 / y.COD_GP_DURATION) - (180 / y.COD_GP_DURATION -
1)

    UNION ALL
    SELECT t.COD_DATE,
        CAST(t.COD_DATE AS BIGINT) * 10000 + t.COD_START_GP COD_DATE_GP,
        t.COD_START_GP,
        t.COD_GP_DURATION,
        FROM_UNIXTIME(UNIX_TIMESTAMP(CAST((CAST(t.COD_DATE AS BIGINT) *
10000 + t.COD_START_GP) AS VARCHAR(20)), 'yyyyMMddHHmm'), 'yyyy-MM-dd HH:mm:ss') DATE_GP
    FROM (SELECT CAST(${hiveconf:COD_DATE} AS INT) COD_DATE,
        COD_START_GP,
        COD_GP_DURATION
    FROM LU_START_GP
        WHERE COD_GP_DURATION = 5
    ) t -- Current day
    ) gp
    WHERE gp.COD_DATE_GP >= evt.MY_START_DATE_GP
    AND gp.COD_DATE_GP <= evt.MY_END_DATE_GP
    ) f
    WHERE f.DURATION > 0;

```


Appendix E.6. AGGREGATION TABLES

AG_LIVE_RATING_DY

```
CREATE TABLE AG_LIVE_RATING_DY
(
  SERVICE_ID VARCHAR(100),
  PROGRAM_ID VARCHAR(100),
  RNK_PROGRAM BIGINT,
  AVG_VIEWERS BIGINT
)
PARTITIONED BY (COD_DATE INT)
LOCATION '/user/hrd/dw/ag_live_rating_gp';
```

Data loading

```
SET MAX_RANK = 100;

ALTER TABLE AG_LIVE_RATING_DY DROP IF EXISTS PARTITION (COD_DATE = ${hiveconf:COD_DATE})
PURGE;

WITH dat
AS (SELECT COD_DATE,
  CASE WHEN GRP_ID > 9 THEN SERVICE_ID ELSE 'GlobalIPTVTVService' END
SERVICE_ID,
  CASE WHEN GRP_ID > 9 THEN CONTENT_ID ELSE '0' END PROGRAM_ID,
  GRP_ID,
  ROUND(AVG(NBR_STBS)) AVG_VIEWERS
FROM (SELECT COD_DATE,
  COD_DATE_GP,
  COD_START_GP,
  COD_GP_DURATION,
  SERVICE_ID,
  CONTENT_ID,
  GROUPING_ID GRP_ID,
  COALESCE(COUNT(DISTINCT STB_ID), 0) NBR_STBS
FROM FACT_EVT_SEGMENTED
WHERE COD_DATE = ${hiveconf:COD_DATE}
AND EVENT_TYPE = 114 -- Program Transition
AND SERVICE_TYPE = 'LIVE' -- Just Live TV
GROUP BY COD_DATE,
  COD_DATE_GP,
  COD_START_GP,
  COD_GP_DURATION,
  SERVICE_ID,
  CONTENT_ID
GROUPING SETS ((COD_DATE, COD_DATE_GP, COD_START_GP, COD_GP_DURATION,
SERVICE_ID, CONTENT_ID),
  (COD_DATE, COD_GP_DURATION))
) f1
GROUP BY COD_DATE,
  SERVICE_ID,
  CONTENT_ID,
  GRP_ID
)
INSERT INTO AG_LIVE_RATING_DY
PARTITION (COD_DATE = ${hiveconf:COD_DATE})
( service_id,
  program_id,
  rnk_program,
  avg_viewers)
SELECT *
FROM (SELECT f.SERVICE_ID,
  f.PROGRAM_ID,
  DENSE_RANK() OVER (PARTITION BY GRP_ID ORDER BY AVG_VIEWERS DESC)
RNK_PROGRAM,
  AVG_VIEWERS
FROM dat f
```

```

        INNER JOIN SS_SERVICE s -- Excluding non-existent TV Channels
        ON (f.SERVICE_ID = s.SERVICE_ID AND
            s.COD_DATE = ${hiveconf:COD_DATE_INV})
        INNER JOIN SS_PROGRAM p -- Excluding non-existent Programs
        ON (f.PROGRAM_ID = p.PROGRAM_ID AND
            p.COD_DATE = ${hiveconf:COD_DATE_INV})
    ) f
WHERE f.RNK_PROGRAM <= ${hiveconf:MAX_RANK};

```

AG_LIVE_REACH_DY

```

CREATE TABLE AG_LIVE_REACH_DY
(
    SERVICE_ID VARCHAR(100),
    RNK_SERVICE BIGINT,
    NBR_VIEWERS BIGINT
)
PARTITIONED BY (COD_DATE INT)
LOCATION '/user/hrd/dw/ag_live_reach_gp';

```

Data loading

```

ALTER TABLE AG_LIVE_REACH_DY DROP IF EXISTS PARTITION (COD_DATE = ${hiveconf:COD_DATE})
PURGE;

INSERT INTO AG_LIVE_REACH_DY
PARTITION (COD_DATE = ${hiveconf:COD_DATE})
( service_id,
  rnk_service,
  nbr_viewers)
SELECT f.SERVICE_ID,
       DENSE_RANK() OVER (PARTITION BY f.GRP_ID ORDER BY f.NBR_VIEWERS DESC) RNK_SERVICE,
       f.NBR_VIEWERS
FROM (SELECT CASE WHEN GROUPING__ID = 1 THEN 'GlobalIPTVTVService' ELSE CHANNEL_ID END
      SERVICE_ID,
       GROUPING__ID GRP_ID,
       COUNT(DISTINCT STB_ID) NBR_VIEWERS
      FROM FACT_ACTIVITY_EVENTS
      WHERE EVENT_TYPE = 100 -- Channel Tune
            AND SERVICE_TYPE = 'LIVE'
            AND ((COD_DATE = ${hiveconf:COD_DATE}) OR
                (COD_DATE = ${hiveconf:COD_DATE_PREV} AND
                 TO_DATE(FROM_UNIXTIME(UNIX_TIMESTAMP(SOURCE_TIMESTAMP) + DURATION, 'yyyy-MM-dd')) >
                 TO_DATE(SOURCE_TIMESTAMP)))
      GROUP BY COD_DATE, CHANNEL_ID
      GROUPING SETS ((COD_DATE, CHANNEL_ID), (COD_DATE))
    ) f;

```

AG_LIVE_SHARE_GP

```

CREATE TABLE AG_LIVE_SHARE_GP
(
    COD_START_GP INT,
    COD_GP_DURATION SMALLINT,
    SERVICE_ID VARCHAR(100),
    NBR_SUBSCRIBERS BIGINT,
    NBR_VIEWERS BIGINT,
    DUR_VIEWING BIGINT
)
PARTITIONED BY (COD_DATE INT)
LOCATION '/user/hrd/dw/ag_live_share_gp';

```

Data loading

```

ALTER TABLE AG_LIVE_SHARE_GP DROP IF EXISTS PARTITION (COD_DATE = ${hiveconf:COD_DATE})
PURGE;

```

```

INSERT INTO AG_LIVE_SHARE_GP
PARTITION (COD_DATE = ${hiveconf:COD_DATE})
( cod_start_gp,
  cod_gp_duration,
  service_id,
  nbr_subscribers,
  nbr_viewers,
  dur_viewing)
SELECT fact.COD_START_GP,
       fact.COD_GP_DURATION,
       CASE WHEN GROUPING__ID = 7 THEN 'GlobalIPTVTVService' ELSE fact.SERVICE_ID END
SERVICE_ID,
       COALESCE(COUNT (DISTINCT stb.SUBSCRIBER_ID), 0) NBR_SUBSCRIBERS,
       COALESCE(COUNT (DISTINCT fact.STB_ID), 0) NBR_VIEWERS,
       COALESCE(SUM(fact.DURATION), 0) DUR_VIEWING
FROM FACT_EVT_SEGMENTED fact
INNER JOIN SS_STB stb
ON (fact.STB_ID = stb.STB_ID AND
    stb.COD_DATE = CAST(${hiveconf:COD_DATE_INV} AS INT)
)
WHERE fact.COD_DATE = CAST(${hiveconf:COD_DATE} AS INT)
      AND fact.EVENT_TYPE = 100 -- Channel Tune Event
      AND fact.SERVICE_TYPE = 'LIVE'
GROUP BY fact.cod_date,
         fact.cod_start_gp,
         fact.cod_gp_duration,
         fact.service_id
GROUPING SETS ((fact.cod_date, fact.cod_start_gp, fact.cod_gp_duration, fact.service_id),
               (fact.cod_date, fact.cod_start_gp, fact.cod_gp_duration));

```

The aggregation tables were created as text files in order to facilitate their export into the RDBMS, even though Apache Sqoop can export tables stored with the ORC file format.

Appendix E.7. TRANSFORMATION PROCESSES DAGs

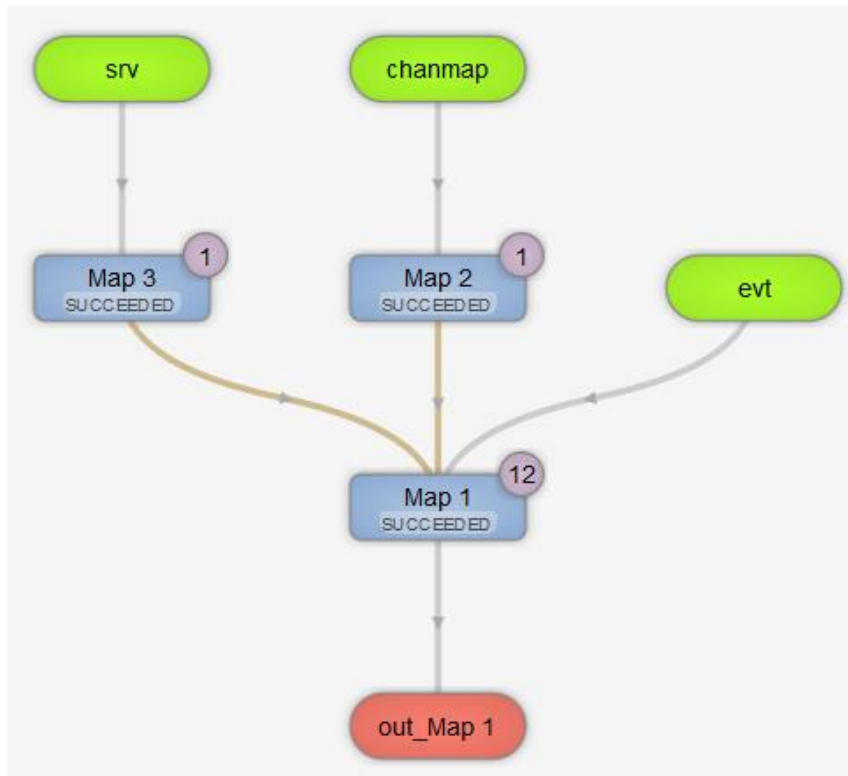


Figure 9.6. *Channel Tune* transformation DAG

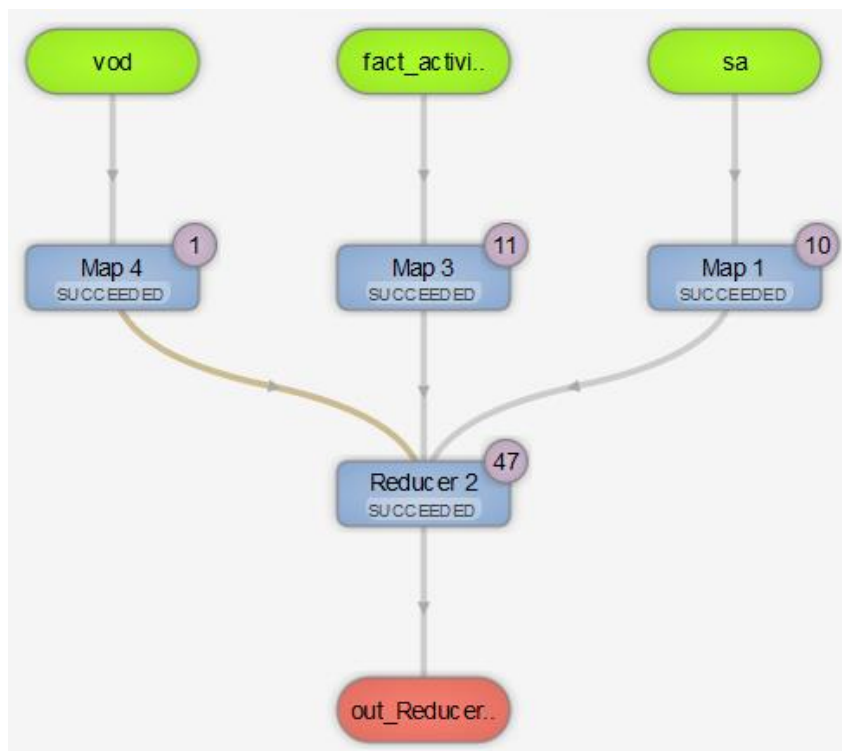


Figure 9.7. *Program Watched* transformation DAG

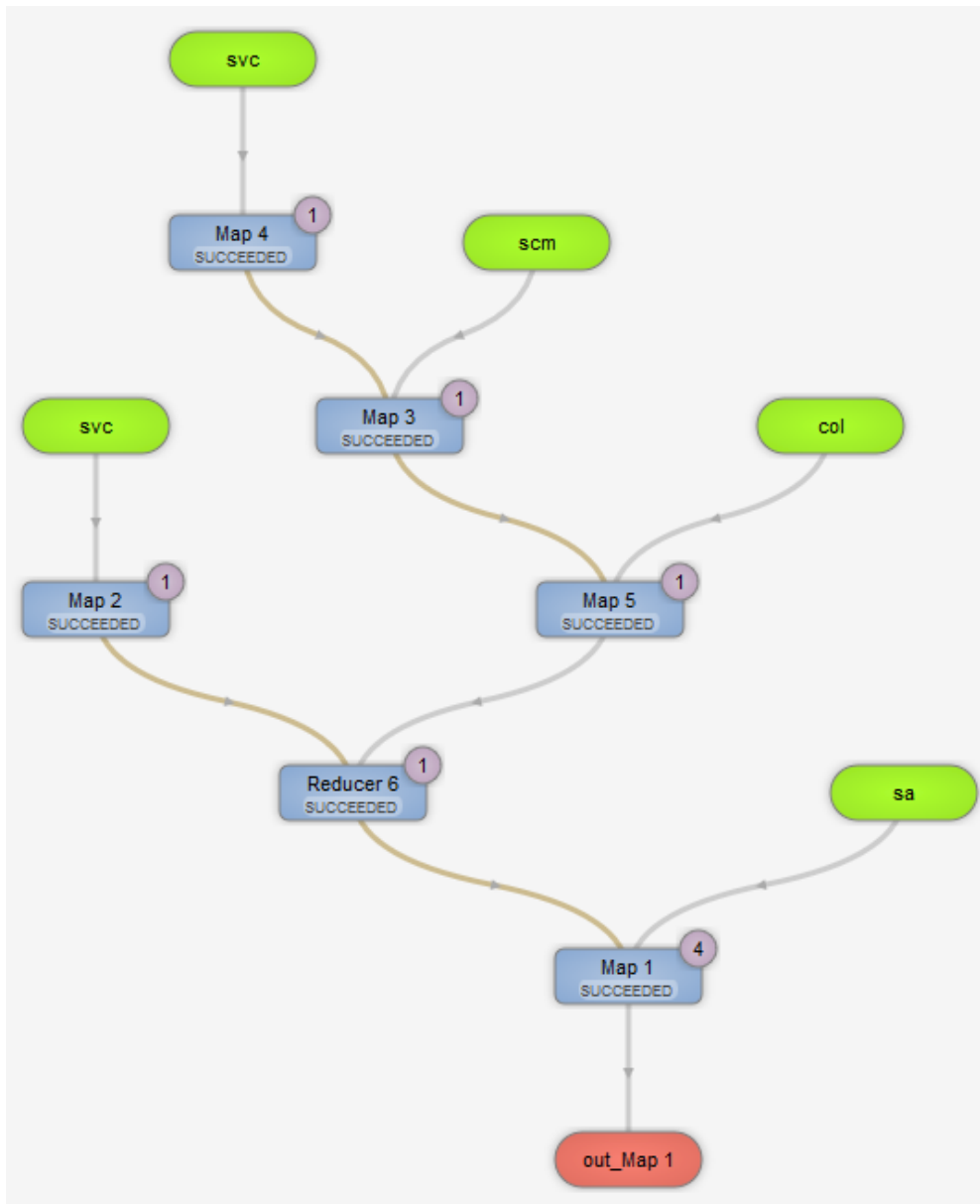


Figure 9.8. DVR Events transformation DAG

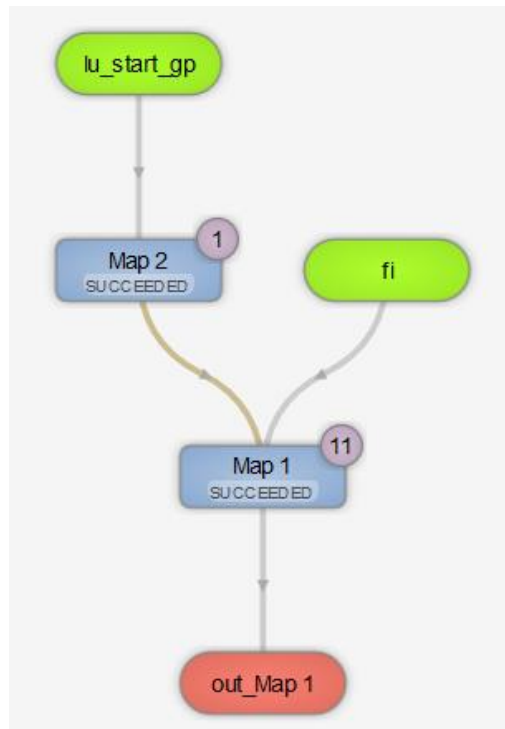


Figure 9.9. *Event Segmentation DAG*

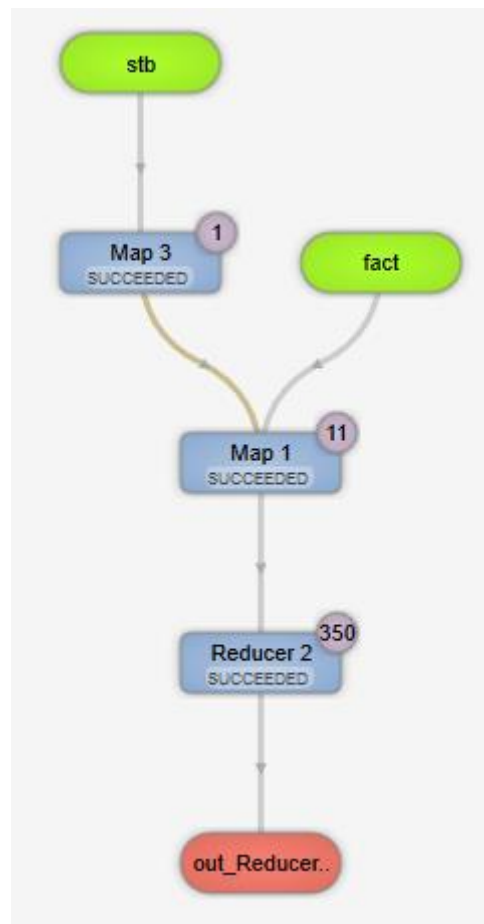


Figure 9.10. *Audiences Aggregation DAG*

Appendix F. PERFORMANCE EXECUTION STATISTICS

This appendix covers the data transformations detailed execution statistics. Note that the 'Avg.' column refers to the average execution time of the three executions that remain after the removal of the best and worst executions. Also, the 'Rows/sec.' column considers the rows in the source table for the first four processes and the number of rows, in the destination table, for the last process, the aggregation.

Appendix F.1. CHANNEL TUNE

Source table		Destination table		Execution time (seconds)						
# Records	MB	# Records	MB	E. #1	E. #2	E. #3	E. #4	E. #5	Avg.	Rows/sec.
1 020 892	200	1 020 892	120	17	16	15	16	16	16	63 806
2 041 947	400	2 041 947	232	30	31	32	31	32	31	65 169
3 062 851	600	3 062 851	352	46	46	48	47	45	46	66 105
4 083 343	800	4 083 343	472	59	58	56	59	58	58	70 000
5 103 459	1 000	5 103 459	584	74	71	70	71	72	71	71 544
6 123 392	1 200	6 123 392	704	86	84	84	83	85	84	72 609
7 143 332	1 400	7 143 332	816	99	99	103	101	101	100	71 196
8 163 314	1 600	8 163 314	936	113	114	113	114	112	113	72 029
9 183 370	1 800	9 183 370	1 088	134	127	130	128	128	129	71 373
10 203 293	2 000	10 203 293	1 216	149	144	143	141	141	143	71 518

Table 9.48. *Channel Tune* execution statistics in the RDBMS

Source table		Destination table		Execution time (seconds)						
# Records	MB	# Records	MB	E. #1	E. #2	E. #3	E. #4	E. #5	Avg.	Rows/sec.
1 020 892	200	1 020 892	41	46	53	58	50	50	51	20 017
2 041 947	400	2 041 947	78	68	66	69	64	73	68	30 177
3 062 851	600	3 062 851	114	72	63	74	66	69	69	44 389
4 083 343	800	4 083 343	149	72	80	93	79	82	80	50 830
5 103 459	1 000	5 103 459	184	103	98	94	99	88	97	52 613
6 123 392	1 200	6 123 392	219	112	124	108	105	123	114	53 557
7 143 332	1 400	7 143 332	254	140	118	141	127	122	130	55 090
8 163 314	1 600	8 163 314	289	126	124	137	135	138	133	61 533
9 183 370	1 800	9 183 370	314	133	135	145	139	151	140	65 752
10 203 293	2 000	10 203 293	360	149	147	150	156	145	149	68 632

Table 9.49. *Channel Tune* execution statistics in Hive

Appendix F.2. PROGRAM WATCHED

Source table		Destination table		Execution time (seconds)						
# Records	MB	# Records	MB	E. #1	E. #2	E. #3	E. #4	E. #5	Avg.	Rows/sec.
1 132 195	200	1 132 195	104	21	20	21	23	22	21	53 072
2 264 301	400	2 264 301	216	46	46	45	46	46	46	49 224

3 396 478	600	3 396 478	336	68	67	67	67	69	67	50 443
4 528 743	800	4 528 743	448	94	93	95	94	92	94	48 350
5 661 056	1 000	5 661 056	560	117	112	114	113	114	114	49 804
6 793 380	1 200	6 793 380	680	129	126	131	125	127	127	53 351
7 925 560	1 400	7 925 560	808	146	145	149	143	144	145	54 659
9 057 613	1 600	9 057 613	920	176	173	177	185	178	177	51 173
10 189 579	1 800	10 189 579	1 088	213	214	204	208	207	209	48 676
11 321 493	2 000	11 321 493	1 216	238	240	237	240	239	239	47 370

Table 9.50. *Program Watched* execution statistics in the RDBMS

Source table		Destination table		Execution time (seconds)						
# Records	MB	# Records	MB	E. #1	E. #2	E. #3	E. #4	E. #5	Avg.	Rows/sec.
1 132 195	200	1 132 195	54	88	84	84	77	79	82	13 751
2 264 301	400	2 264 301	105	84	92	93	91	90	91	24 882
3 396 478	600	3 396 478	156	109	109	111	106	123	110	30 971
4 528 743	800	4 528 743	208	120	112	118	125	114	117	38 597
5 661 056	1 000	5 661 056	259	139	134	137	132	148	137	41 422
6 793 380	1 200	6 793 380	314	175	166	149	140	146	154	44 209
7 925 560	1 400	7 925 560	366	164	167	192	162	164	165	48 034
9 057 613	1 600	9 057 613	418	181	185	287	193	175	186	48 610
10 189 579	1 800	10 189 579	470	187	186	185	192	244	188	54 104
11 321 493	2 000	11 321 493	521	189	188	191	213	191	190	59 482

Table 9.51. *Program Watched* execution statistics in Hive

Appendix F.3. DVR EVENTS

Source table		Destination table		Execution time (seconds)						
# Records	MB	# Records	MB	E. #1	E. #2	E. #3	E. #4	E. #5	Avg.	Rows/sec.
1 000 000	201	1 000 000	128	18	15	16	16	15	16	63 830
2 000 000	402	2 000 000	224	31	31	31	30	31	31	64 516
3 000 000	603	3 000 000	320	50	48	46	45	44	46	64 748
4 000 000	804	4 000 000	424	61	59	62	60	61	61	65 934
5 000 000	1 005	5 000 000	528	75	75	75	76	78	75	66 372
6 000 000	1 206	6 000 000	616	84	85	86	86	86	86	70 039
7 000 000	1 407	7 000 000	720	101	100	100	98	98	99	70 470
8 000 000	1 608	8 000 000	808	114	112	112	116	112	113	71 006
9 000 000	1 809	9 000 000	920	131	127	128	129	128	128	70 130
10 000 000	2 010	10 000 000	1 008	142	140	140	140	141	140	71 259

Table 9.52. *DVR Events* transformation execution statistics in the RDBMS

Source table		Destination table		Execution time (seconds)						
# Records	MB	# Records	MB	E. #1	E. #2	E. #3	E. #4	E. #5	Avg.	Rows/sec.
1 000 000	201	1 000 000	34	42	44	41	41	43	42	23 810
2 000 000	402	2 000 000	68	58	54	46	52	49	52	38 710
3 000 000	603	3 000 000	102	53	74	55	56	71	61	49 451
4 000 000	804	4 000 000	136	57	97	64	65	68	66	60 914
5 000 000	1 005	5 000 000	169	69	67	80	98	87	79	63 559
6 000 000	1 206	6 000 000	203	96	109	97	92	92	95	63 158
7 000 000	1 407	7 000 000	237	107	112	104	108	99	106	65 831
8 000 000	1 608	8 000 000	271	118	118	110	111	109	113	70 796
9 000 000	1 809	9 000 000	305	117	116	116	117	120	117	77 143
10 000 000	2 010	10 000 000	339	129	139	128	119	126	128	78 329

Table 9.53. *DVR Events* transformation execution statistics in Hive

Appendix F.4. EVENT SEGMENTATION

Source table		Destination table		Execution time (seconds)						
# Records	MB	# Records	MB	E. #1	E. #2	E. #3	E. #4	E. #5	Avg.	Rows/sec.
1 020 892	120	7 003 222	488	116	112	110	114	112	113	62 159
2 041 947	232	14 382 135	1 000	203	199	200	199	204	201	71 672
3 062 851	352	21 295 751	1 536	294	293	295	289	294	294	72 517
4 083 343	472	28 282 050	1 984	366	368	366	371	369	368	76 923
5 103 459	584	35 766 254	2 496	446	444	445	445	449	445	80 313
6 123 392	704	43 472 987	3 008	529	537	541	528	534	533	81 512
7 143 332	816	51 199 355	3 520	619	615	617	621	638	619	82 713
8 163 314	936	58 820 755	4 096	695	705	706	710	717	707	83 198
9 183 370	1 088	66 387 314	4 608	781	787	780	793	828	787	84 355
10 203 293	1 216	73 915 173	5 120	841	855	842	850	864	849	87 061

Table 9.54. *Event Segmentation* execution statistics in the RDBMS

Source table		Destination table		Execution time (seconds)						
# Records	MB	# Records	MB	E. #1	E. #2	E. #3	E. #4	E. #5	Avg.	Rows/sec.
1 020 892	120	7 003 222	81	163	159	171	167	162	164	42 703
2 041 947	232	14 382 135	162	183	187	199	200	197	194	74 008
3 062 851	352	21 295 751	238	253	190	208	244	227	226	94 090
4 083 343	472	28 282 050	314	230	252	249	225	273	244	116 069
5 103 459	584	35 766 254	393	315	304	302	320	372	313	114 269
6 123 392	704	43 472 987	475	413	345	396	360	346	367	118 348
7 143 332	816	51 199 355	557	448	423	398	445	425	431	118 792
8 163 314	936	58 820 755	638	494	425	444	456	486	462	127 318
9 183 370	1 088	66 387 314	719	501	484	471	497	524	494	134 387
10 203 293	1 216	73 915 173	800	575	517	517	548	530	532	139 025

Table 9.55. *Event Segmentation* execution statistics in Hive

Appendix F.5. AUDIENCES AGGREGATION

Source table		Destination table		Execution time (seconds)						
# Records	MB	# Records	MB	E. #1	E. #2	E. #3	E. #4	E. #5	Avg.	Rows/sec.
7 003 222	488	41 215	8	94	91	92	91	92	92	76 399
14 382 135	1 000	54 979	8	233	204	208	203	202	205	70 157
21 295 751	1 536	62 332	8	400	395	365	380	374	383	55 602
28 282 050	1 984	66 730	8	647	588	714	602	708	652	43 355
35 766 254	2 496	67 361	8	840	897	851	910	868	872	41 016
43 472 987	3 008	67 618	8	1 199	932	1 170	937	1 186	1 098	39 605
51 199 355	3 584	67 730	8	1 306	1 303	1 308	1 306	1 322	1 307	39 183
58 820 755	4 096	67 982	8	1 483	1 529	1 491	1 536	1 496	1 505	39 075
66 387 314	4 608	68 086	8	1 762	1 662	1 934	1 651	1 814	1 746	38 023
73 915 173	5 120	68 253	8	1 933	2 052	1 896	2 070	1 848	1 960	37 705

Table 9.56. *Audiences Aggregation* execution statistics in the RDBMS

Source table		Destination table		Execution time (seconds)						
# Records	MB	# Records	MB	E. #1	E. #2	E. #3	E. #4	E. #5	Avg.	Rows/sec.
7 003 222	488	41 215	1	144	152	136	141	176	146	48 077
14 382 135	1 000	54 979	1	210	202	261	218	166	210	68 486
21 295 751	1 536	62 332	1	301	242	286	270	247	268	79 561
28 282 050	1 984	66 730	1	404	350	331	380	342	357	79 148
35 766 254	2 496	67 361	1	386	388	414	400	415	401	89 267
43 472 987	3 008	67 618	2	431	458	467	474	495	466	93 223
51 199 355	3 584	67 730	1	507	495	617	481	469	494	103 573
58 820 755	4 096	67 982	3	590	550	525	560	516	545	107 928
66 387 314	4 608	68 086	1	625	619	642	621	590	622	106 789
73 915 173	5 120	68 253	1	807	734	742	691	724	733	100 793

Table 9.57. *Audiences Aggregation* execution statistics in Hive

Appendix G. SCALABILITY EXECUTION STATISTICS

This appendix contains the execution statistics of the scaled systems. Note that the first three scenarios of the tests, performed in Hive, are using the results previously collected before the cluster was scaled-out. Only with the fourth scenario we can start experiencing performance benefits, since it is at this stage that the executions start to use at least four tasks and, therefore, possibly the full capabilities of the cluster.

Appendix G.1. CHANNEL TUNE

Source table		Destination table		Execution time (seconds)						
# Records	MB	# Records	MB	E. #1	E. #2	E. #3	E. #4	E. #5	Avg.	Rows/sec.
1 020 892	200	1 020 892	41	46	53	58	50	50	51	20 017
2 041 947	400	2 041 947	78	68	66	69	64	73	68	30 177
3 062 851	600	3 062 851	114	72	63	74	66	69	69	44 389
4 083 343	800	4 083 343	149	69	79	60	70	61	67	61 250
5 103 459	1 000	5 103 459	184	79	84	79	77	77	78	65 151
6 123 392	1 200	6 123 392	219	98	86	87	86	94	89	68 802
7 143 332	1 400	7 143 332	254	109	99	93	97	106	101	70 960
8 163 314	1 600	8 163 314	289	114	112	116	121	111	114	71 608
9 183 370	1 800	9 183 370	314	129	126	125	129	122	127	72 500
10 203 293	2 000	10 203 293	360	130	131	132	135	135	133	76 909

Table 9.58. *Channel Tune* execution statistics in Hive (scaled-out)

Appendix G.2. EVENT SEGMENTATION

Source table		Destination table		Execution time (seconds)						
# Records	MB	# Records	MB	E. #1	E. #2	E. #3	E. #4	E. #5	Avg.	Rows/sec.
1 020 892	120	7 003 222	488	110	108	109	108	108	108	64 645
2 041 947	232	14 382 135	1 000	199	198	200	199	202	199	72 151
3 062 851	352	21 295 751	1 536	290	270	274	272	279	275	77 439
4 083 343	472	28 282 050	1 984	348	351	346	349	354	349	80 960
5 103 459	584	35 766 254	2 496	443	453	446	448	441	446	80 253
6 123 392	704	43 472 987	3 008	531	521	531	527	534	530	82 076
7 143 332	816	51 199 355	3 520	611	604	602	610	609	608	84 256
8 163 314	936	58 820 755	4 096	682	684	681	680	689	682	86 205
9 183 370	1 088	66 387 314	4 608	787	789	797	781	794	790	84 035
10 203 293	1 216	73 915 173	5 120	846	848	853	841	843	846	87 405

Table 9.59. *Event Segmentation* execution statistics in the RDBMS (scaled-up)

Source table		Destination table		Execution time (seconds)						
# Records	MB	# Records	MB	E. #1	E. #2	E. #3	E. #4	E. #5	Avg.	Rows/sec.
1 020 892	120	7 003 222	81	163	159	171	167	162	164	42 703
2 041 947	232	14 382 135	162	183	187	199	200	197	194	74 008

3 062 851	352	21 295 751	238	253	190	208	244	227	226	94 090
4 083 343	472	28 282 050	314	229	209	238	209	215	218	129 933
5 103 459	584	35 766 254	393	282	276	275	264	251	272	131 655
6 123 392	704	43 472 987	475	332	341	295	328	314	325	133 900
7 143 332	816	51 199 355	557	398	411	411	323	318	377	135 687
8 163 314	936	58 820 755	638	445	419	417	455	428	431	136 581
9 183 370	1 088	66 387 314	719	469	455	459	468	470	465	142 666
10 203 293	1 216	73 915 173	800	502	499	507	409	519	503	147 046

Table 9.60. *Event Segmentation* execution statistics in Hive (scaled-out)

Appendix G.3. AUDIENCES AGGREGATION

Source table		Destination table		Execution time (seconds)						
# Records	MB	# Records	MB	E. #1	E. #2	E. #3	E. #4	E. #5	Avg.	Rows/sec.
7 003 222	488	41 215	8	88	88	89	91	88	88	79 282
14 382 135	1 000	54 979	8	185	183	197	184	186	185	77 741
21 295 751	1 536	62 332	8	263	258	261	258	264	261	81 697
28 282 050	1 984	66 730	8	356	360	360	359	364	360	78 634
35 766 254	2 496	67 361	8	503	471	470	481	483	478	74 773
43 472 987	3 008	67 618	8	638	631	608	617	624	624	69 668
51 199 355	3 584	67 730	8	895	804	867	823	840	843	60 711
58 820 755	4 096	67 982	8	898	1 017	949	917	1 012	959	61 314
66 387 314	4 608	68 086	8	1 167	1 135	1 193	1 187	1 158	1 171	56 709
73 915 173	5 120	68 253	8	1 224	1 359	1 216	1 243	1 286	1 251	59 085

Table 9.61. *Audiences Aggregation* execution statistics in the RDBMS (scaled-up)

Source table		Destination table		Execution time (seconds)						
# Records	MB	# Records	MB	E. #1	E. #2	E. #3	E. #4	E. #5	Avg.	Rows/sec.
7 003 222	488	41 215	1	144	152	136	141	176	146	48 077
14 382 135	1 000	54 979	1	210	202	261	218	166	210	68 486
21 295 751	1 536	62 332	1	301	242	286	270	247	268	79 561
28 282 050	1 984	66 730	1	249	255	242	235	299	249	113 735
35 766 254	2 496	67 361	1	299	340	301	308	329	313	114 391
43 472 987	3 008	67 618	2	404	375	348	377	407	385	112 819
51 199 355	3 584	67 730	1	451	450	482	466	447	456	112 361
58 820 755	4 096	67 982	3	521	517	564	512	533	524	112 325
66 387 314	4 608	68 086	1	571	604	629	566	544	580	114 395
73 915 173	5 120	68 253	1	647	612	628	625	686	633	116 708

Table 9.62. *Audiences Aggregation* execution statistics in Hive (scaled-out)

Appendix H. VISUALIZATION

In this appendix we extend the visualization layer presented in section 5.6.1 with a set of reports, developed on top of the produced aggregation tables, with the purpose of communicating visually the insights gathered from the raw data originated in Mediaroom. These reports are merely examples of what we can extract from the information generated by our data warehouse. For demonstration purposes, we are focusing our attention around some of the best-known metrics in the field of television audiences, like the *Reach*, the *Rating* and the *Share* of Live television, but not to focus solely on Live TV, we also present some simple measurements related to Video-on-Demand.

One of the most relevant television metrics is the *Share* of each channel. In Figure 9.11 we present a report that shows the *Share* of the top channels throughout an entire day, in slices of five minutes. The calculation of the *Share* here is the percentage of viewers, of a given channel, during each slot of five minutes and the percentage considers, as universe, only the people that were watching live television in each five-minute period. Using information as granular as this can help us determine, for example, the time slots where advertising would be exposed to the largest number of viewers. From this report, and on a more specific view, we can observe that the channel that has the most *Share* throughout the day is not the one that leads during a period of about two hours. By relating channel and program information we could assess that during the mentioned period, a football match was being broadcasted and from there we can even understand, in a visual manner, the period that reflects the half-time where many viewers navigated away from the channel.

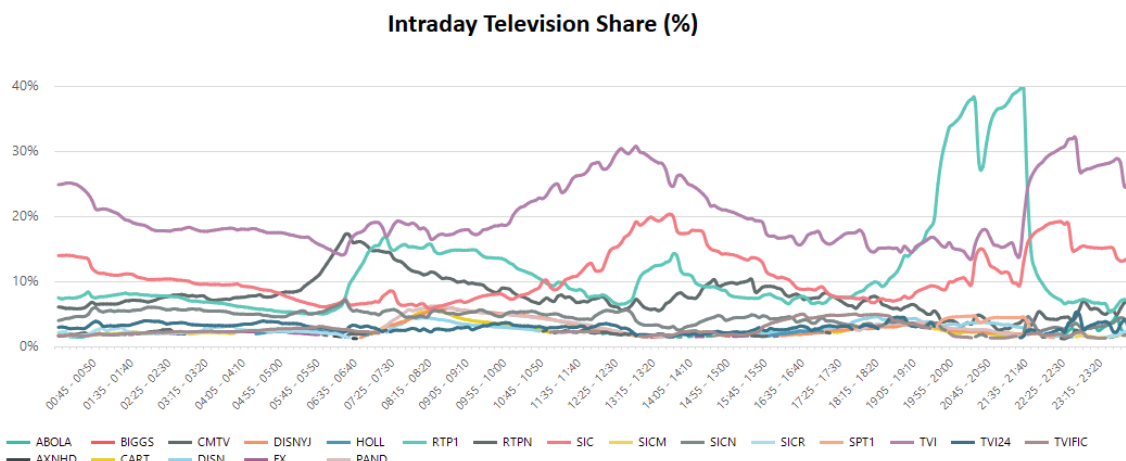


Figure 9.11. *Intraday Television Share (%)* report

The next report, presented in Figure 9.12, shows the daily *Reach* of the top ten channels during a chosen set of days. In this report, the metric *Reach* consists of the percentage of individuals, from the entire population, that visualized any given channel at least once during the entire day. The calculation of *Share* considers as population just the customers utilizing the Live Television service, whereas the *Reach* considers as its universe the entire population of the IPTV service, no matter if they are active or not during the analysis period.

Television Channel Reach (%)

Day	2016-05-03		2016-05-04		2016-05-05		Average
Channel	Reach (%)	Rank	Reach (%)	Rank	Reach (%)	Rank	Reach (%)
TVI	37.82%	1	48.33%	1	53.60%	1	46.58%
SIC	25.76%	2	36.13%	3	46.55%	2	36.15%
RTP1	20.40%	3	47.83%	2	35.23%	3	34.49%
CMTV	18.20%	4	29.49%	4	34.76%	4	27.48%
SICN	13.47%	5	20.63%	5	24.71%	5	19.60%
TVI24	10.51%	6	19.83%	6	21.67%	6	17.34%
RTPN	7.82%	7	13.76%	7	16.70%	7	12.76%
SICR	5.59%	9	9.43%	10	12.55%	8	9.19%
ABOLA	6.13%	8	10.42%	8	10.81%	10	9.12%
TVIFIC	5.50%	10	9.79%	9	11.20%	9	8.83%

Figure 9.12. Television Channel Reach (%) report

Moving to a lower granularity level, the television program, we are able to capture and deliver the *Rating* of each program for any given day. Figure 9.13 presents us the top ten programs with the highest rating during an entire day. The *Rating* is calculated here as a percentage of the average population that watched the program during its broadcast time. Like the *Reach*, the calculation of the *Rating* considers as its universe the entire population of the IPTV service. This report also includes two more visual components the quickly communicate to the user which are the channels broadcasting the programs with the highest *Ratings* and which percentage of the population watched, on average, these programs.

Daily Program Rating (%)

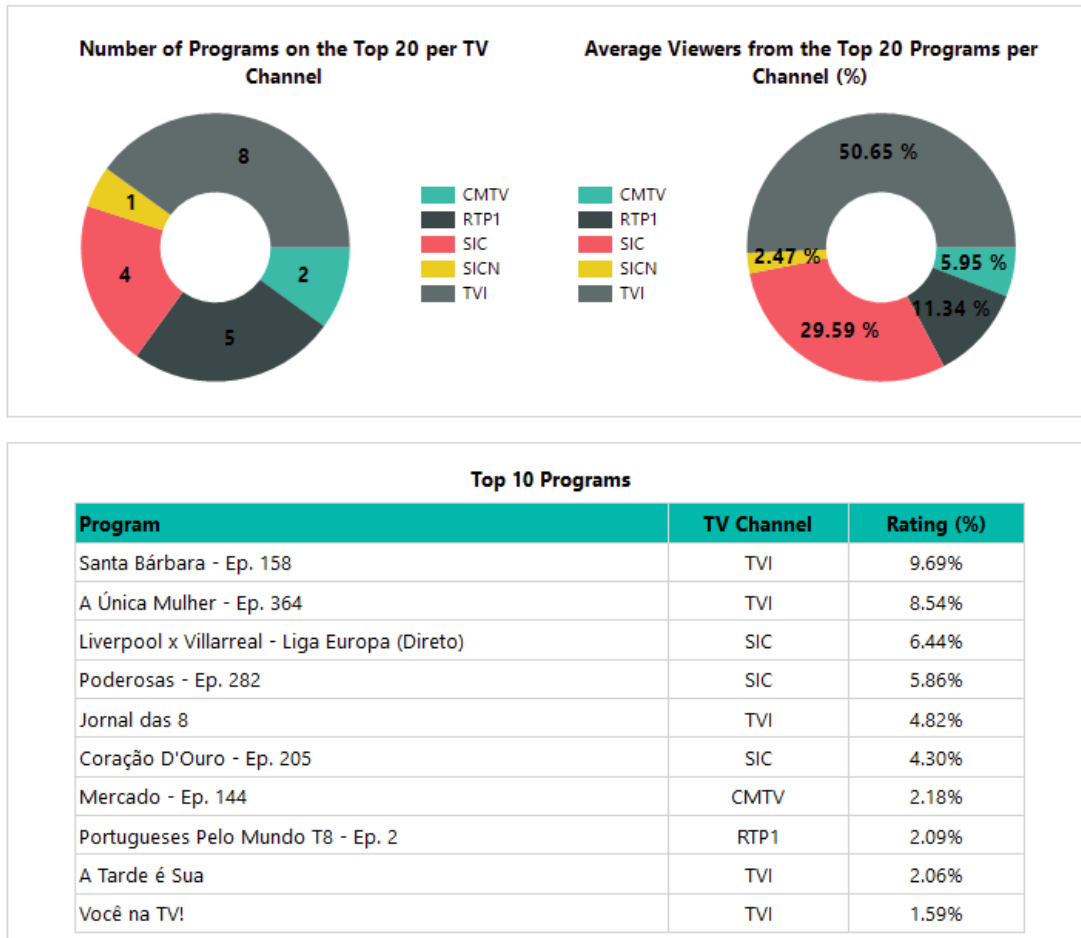


Figure 9.13. Daily Program Rating (%) report

Understanding the behaviors of the IPTV service can also be done using other dimensions beyond Live Television. In Figure 9.14 we depict an example of how we can use the raw data from the Mediaroom platform to capture insights related to the visualization of Video-on-Demand, by their genre and video provider, during different days of the week.

Weekly VoD Visualizations by Genre and Provider

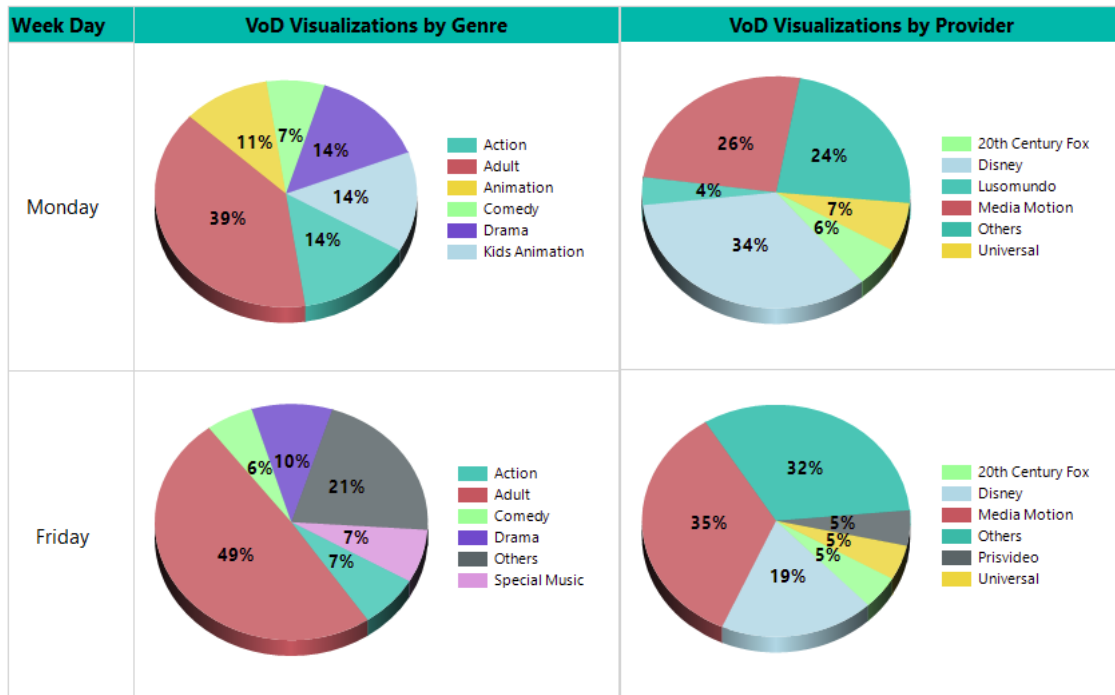


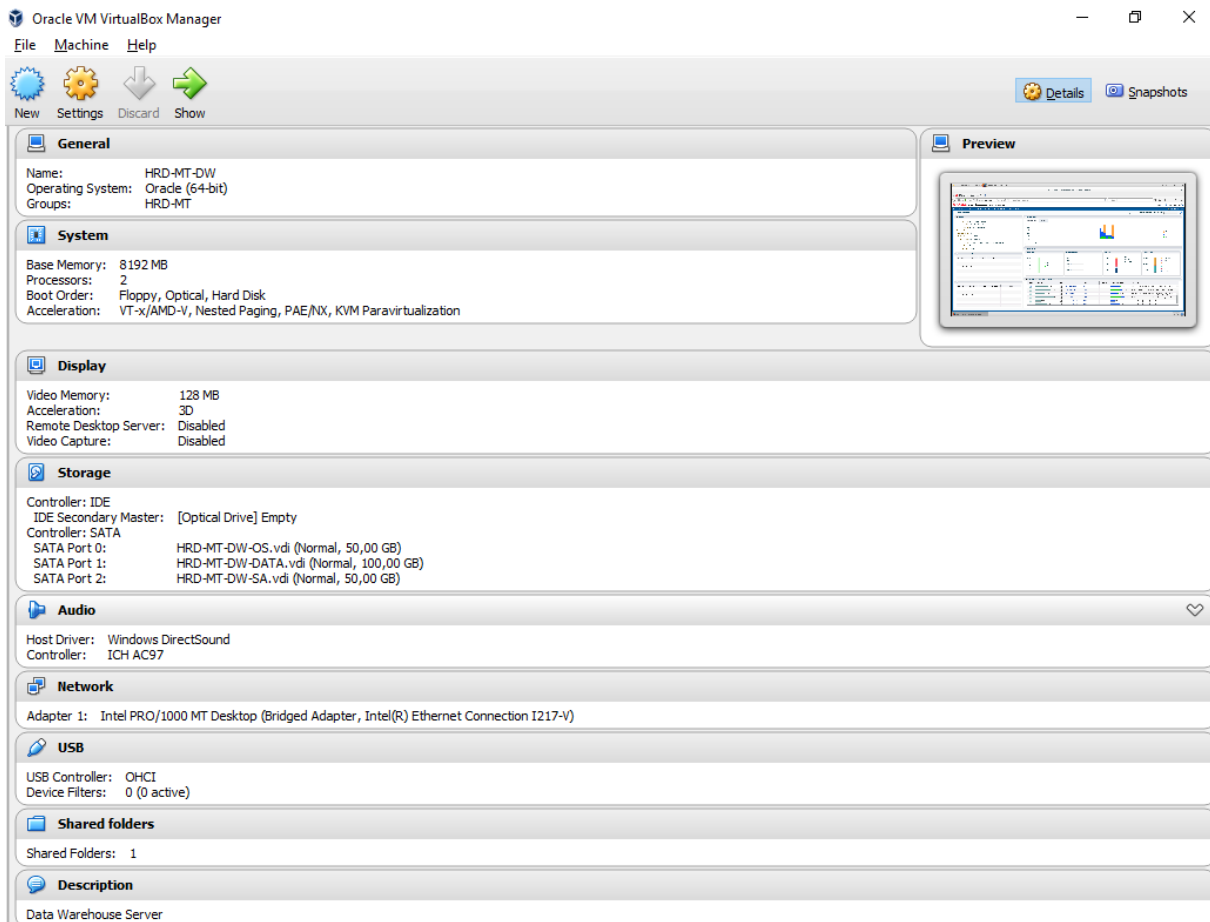
Figure 9.14. *Weekly VoD Visualizations* report

10.ANNEXES

Annex A. RDBMS INSTALLATION

This annex covers the installation of a virtual machine that supports a database instance using Oracle 12c on Oracle Linux Server 7.2 under Oracle VirtualBox 5.1.18.

Annex A.1. VIRTUAL MACHINE CONFIGURATION



Annex A.2. OPERATING SYSTEM INSTALLATION

Hostname: hrd-mt-dw.v.nexus

Manual network configuration to have the exact desired IPs (configure -> IPv4 Settings):

Address: 192.168.1.100

Netmask: 255.255.255.0

Default gateway: 192.168.1.254

DNS: 8.8.8.8, 4.4.4.4

Software selection

Base Environment: Server with GUI

Add-Ons for selected environment: none selected

Disk partitioning

Manual to add a swap partition of 10GB

After the installation perform the update of the operating system:

```
$ su -  
$ yum update
```

Install the VirtualBox Guest Additions. This requires the installation of the kernel development packages:

```
$ su -  
$ yum install kernel-uek-devel-$(uname -r)
```

Now insert the Guest Additions CD Image and install it. After its completion, reboot.

Update /etc/hosts so that the database host knows the other machines in the network:

```
$ su -  
$ gedit /etc/hosts
```

Contents of /etc/hosts:

```
192.168.1.100    hrd-mt-dw.v.nexus  hrd-mt-dw  
192.168.1.101    hrd-mt-h01.v.nexus hrd-mt-h01  
192.168.1.102    hrd-mt-h02.v.nexus hrd-mt-h02  
192.168.1.103    hrd-mt-h03.v.nexus hrd-mt-h03  
192.168.1.104    hrd-mt-h04.v.nexus hrd-mt-h04
```

For this exercise, we will disable the firewall to facilitate the communication between the different machines in the network:

```
$ su -  
$ systemctl stop firewalld  
$ systemctl disable firewalld
```

At this point the operating system is ready for the installation of the database software.

Hostname: hrd-mt-dw

IP: 192.168.1.100

Annex A.3. OPERATING SYSTEM CONFIGURATION

After the installation of the operating system we need to proceed with the configuration of several parameters before we can install the database. Before installing Oracle 12c itself, there is a set of pre-requisites that should be validated, and some configurations that need to be assured. In order to facilitate this step, the following package was used: “How I Simplified Oracle Database 12c and 11g Installations on Oracle Linux 6”²¹.

```
$ su -  
$ yum install oracle-rdbms-server-12cR1-preinstall
```

Another important step that must be performed, before installing Oracle, is the creation of the users:

```
$ su -  
$ /usr/sbin/groupadd oinstall  
$ /usr/sbin/groupadd dba  
$ /usr/sbin/useradd -g oinstall -G dba oracle  
$ passwd oracle
```

The operating system parameter definition is related to the system resources and their correct setting is important to maximize the database performance.

Parameter definition in ‘/etc/sysctl.conf’:

```
fs.aio-max-nr = 1048576  
fs.file-max = 6815744  
kernel.shmall = 838860  
kernel.shmmax = 4294967296  
kernel.shmmni = 4096  
kernel.sem = 250 32000 100 128  
net.ipv4.ip_local_port_range = 9000 65500  
net.core.rmem_default = 262144  
net.core.rmem_max = 4194304  
net.core.wmem_default = 262144  
net.core.wmem_max = 1048586
```

For ‘kernel.shmall’ Oracle recommends to be 40% of the physical memory in pages: 8GB * 40% -> 3,2GB * 1014 * 1024 * 1024 / 4096 (page size). For ‘kernel.shmmax’ Oracle recommends setting it to half of the physical memory in bytes. The other parameters are correctly set by the execution of the pre-requisites package.

After making the changes in the file ‘/etc/sysctl.conf’, to make them active reboot the system or perform the following command:

```
$ /sbin/sysctl -p
```

²¹ <http://www.oracle.com/technetwork/articles/servers-storage-admin/ginnydbinstallonlinux-488779.html>
(Accessed on 2016-10-13)

With the use of the 'ulimit' command we need to reconfigure some system-wide specific limits:

```
$ su -
$ vi /etc/security/limits.conf

oracle soft   nofile 4096
oracle hard   nofile 65536
oracle soft   nproc 2047
oracle hard   nproc 16384
oracle soft   stack 10240
oracle hard   stack 32768
oracle soft   memlock 7633635
oracle hard   memlock 7633635
```

The definition of these limits should also be present in the shell environment of the user that owns and executes the database instance (oracle). So, the following commands are to be added to the user oracle '.bashrc' file:

```
ulimit -Hn 65536
ulimit -Sn 4096
ulimit -Hu 16384
ulimit -Su 2047
ulimit -Hs 32768
ulimit -Ss 10240
```

During the configuration of the Oracle instance, if the MEMORY_TARGET feature is used, we will need to define the system available shared memory to be at least the value defined by the MEMORY_TARGET parameter²². To allow this, we will increase the available shared memory and set it to 6 gigabytes:

```
$ su -
$ umount tmpfs
$ mount -t tmpfs shmfs -o size=6G /dev/shm
```

To make this change permanent add the following command to the file /etc/fstab:

```
tmpfs /dev/shm tmpfs defaults,size=6G 0 0
```

To make this change active reboot the system or execute:

```
$ umount tmpfs
$ mount -a
```

The final step, before we can start the installation of the database software and the configuration of the instance, is the creation of the directories that will support the software and the data.

²² <http://oraclesivaram.blogspot.pt/2015/09/ins-35172-target-database-memory-xxxmb.html> (Accessed on 2016-10-13)

Software:

```
$ su -  
$ mkdir -p /app/  
$ chown -R oracle:oinstall /app/  
$ chmod -R 775 /app/
```

Data:

```
$ su -  
$ mkdir -p /mnt/sdb/oradata/  
$ chown -R oracle:oinstall /mnt/sdb/oradata/  
$ chmod -R 775 /mnt/sdb/oradata/
```

At this point we can proceed with the database installation since all the requirements are met.

Annex A.4. DATABASE INSTALLATION

With the user that will be owner the installation (oracle) execute the installer:

```
$ . runInstaller
```

During the installation wizard, the following options were selected:

Install Option: Create and configure a database

System Class: Server class

Grid Installation Options: Single instance database installation

Install Type: Advanced install

Database Edition: Enterprise Edition

Configuration Type: Data Warehousing

Database identifiers:

Global database name: **dw.v.nexus**

SID: **dw**

Uncheck the “Create as Container database”

Configuration Options:

Memory: 5120 (~66% of 8GB) and enable Automatic Memory Management

Sample schemas: uncheck ‘Create database with sample schema’

Database storage: Filesystem. Change to match the folder on /mnt/sdb/oradata

Operating System Groups: all dba

Annex B. HADOOP CLUSTER INSTALLATION

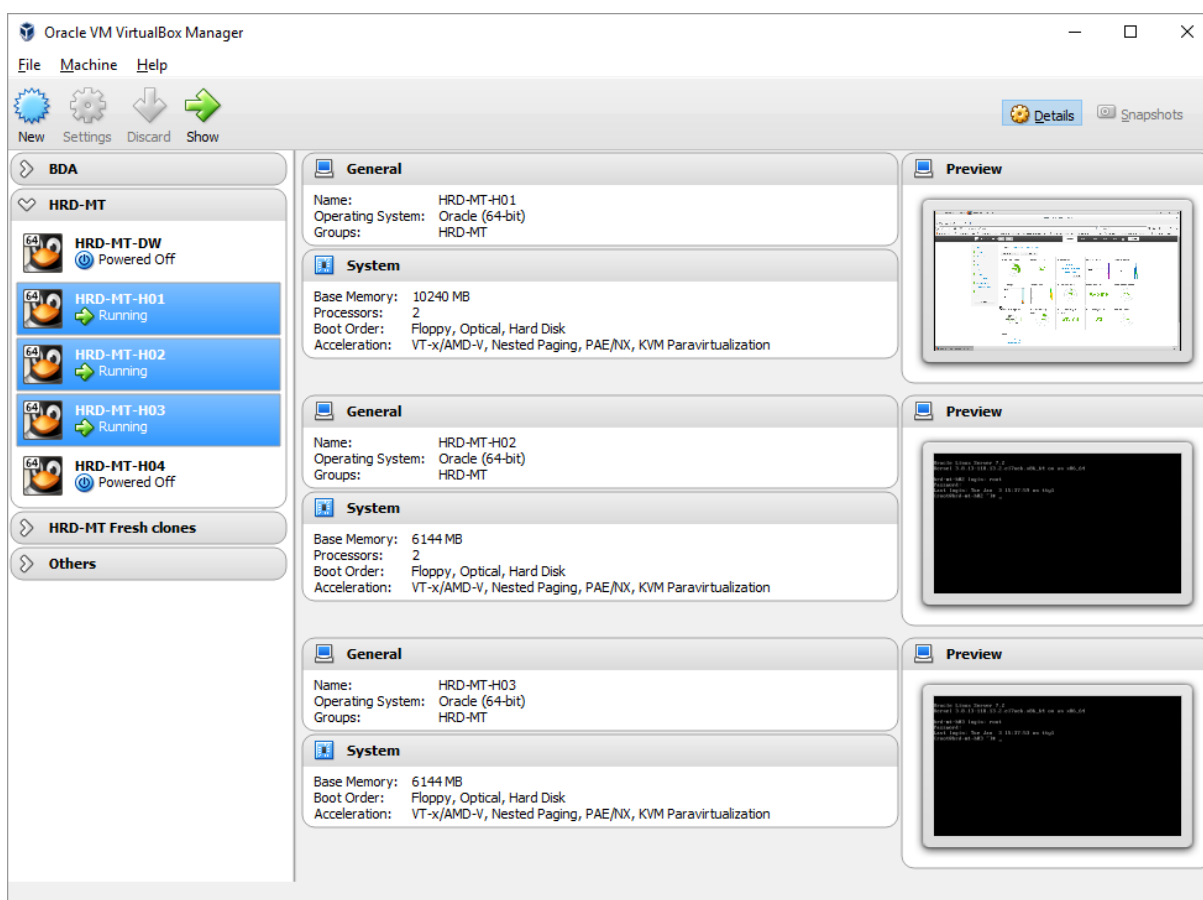
This annex guide us through the installation of a Hadoop cluster using Hortonworks Data Platform 2.5, in a virtual environment manage by Oracle VirtualBox 5.1.18.

Annex B.1. VIRTUAL MACHINE CONFIGURATION

The resources available on the host computer are limited and, therefore, we need to distribute them, among the nodes, carefully. The table below shows us the nodes that will be part of our cluster.

Hostname	IP Address	Functions	Memory	# CPUs	Operating System
hrd-mt-h01.v.nexus	192.168.1.101	NameNode DataNode	10 240 MB	2	Oracle Linux 7.2
hrd-mt-h03.v.nexus	192.168.1.102	DataNode	6 144 MB	2	Oracle Linux 7.2
hrd-mt-h04.v.nexus	192.168.1.103	DataNode	6 144 MB	1	Oracle Linux 7.2
hrd-mt-h04.v.nexus	192.168.1.104	DataNode	6 144 MB	1	Oracle Linux 7.2

Due to the limitation of the host's resources, only the first node, that will be accumulating the functions of both DataNode and NameNode, will have installed a graphic user interface for the sake of facilitating some management operations.



Above is the configuration of the virtual machines hosting our cluster's nodes. The network interfaces for the virtual machines are configured as 'bridged network' so that all the nodes are easily accessible

throughout the network, not only amongst themselves but also to any other computers that might want to connect directly to them.

Annex B.2. OPERATING SYSTEM INSTALLATION

The installation of the operating systems and their configurations is divided in two different steps, aligned with the different configurations of the virtual machines. In the first step, we will install the main node that will accumulate several roles and from where we will coordinate the cluster deployment, and in the second step we will install the other nodes mostly devoted to the roles of DataNode and NodeMaster.

Installation of hrd-mt-h01

Manual **network configuration** to have the exact desired IPs (Configure -> IPv4 Settings):

Address: 192.168.1.101

Netmask: 255.255.255.0

Default gateway: 192.168.1.254

DNS: 8.8.8.8, 4.4.4.4

Component installation:

Base Environment: Server with GUI

Add-Ons for selected environment: None selected

Installation of hrd-mt-h02

Manual **network configuration** to have the exact desired IPs (Configure -> IPv4 Settings):

Address: 192.168.1.102

Netmask: 255.255.255.0

Default gateway: 192.168.1.254

DNS: 8.8.8.8, 4.4.4.4

Component installation:

Base Environment: Minimum install

Add-Ons for selected environment: None selected

There is no need to install the remaining nodes since they'll be generated by cloning the hard disk of hrd-mt-h02 after its configuration is finalized.

Annex B.3. OPERATING SYSTEM CONFIGURATION

Configuration of hrd-mt-h01

After the installation perform the update of the operating system:

```
$ su -  
$ yum update
```

Install the VirtualBox Guest Additions. This requires the installation of the kernel development packages:

```
$ su -  
$ yum install kernel-uek-devel-$(uname -r)
```

Now insert the Guest Additions CD Image and install it. After its completion reboot.

Update /etc/hosts so that the database host knows the other machines in the network:

```
$ su -  
$ gedit /etc/hosts
```

Contents of /etc/hosts:

192.168.1.100	hrd-mt-dw.v.nexus	hrd-mt-dw
192.168.1.101	hrd-mt-h01.v.nexus	hrd-mt-h01
192.168.1.102	hrd-mt-h02.v.nexus	hrd-mt-h02
192.168.1.103	hrd-mt-h03.v.nexus	hrd-mt-h03
192.168.1.104	hrd-mt-h04.v.nexus	hrd-mt-h04

For this exercise, we will disable the firewall to facilitate the communication between the different machines in the network:

```
$ su -  
$ systemctl stop firewalld  
$ systemctl disable firewalld
```

In order to deploy the cluster via Ambari, the Hadoop management tool used by Hortonworks Data Platform, we need to install the Network Time Protocol Daemon service (NTPD).

Install NTP service:

```
$ su -  
$ yum install ntp
```

Start the NTPD service and set it to launch automatically upon boot:

```
$ service ntpd start  
$ chkconfig ntpd on
```

Ambari, the tool responsible for the cluster installation and deployment, recommends the following maximum open file descriptors configuration:

```
$ ulimit -Sn
$ ulimit -Hn

# if the output is not greater than 10000, run the following command:
$ ulimit -n 10000
```

The definition of these limits should also be present in the shell environment of the user that will execute Ambari. So, the following command is to be added to the '.bashrc' file:

```
ulimit -n 10000
```

Configuration of hrd-mt-h02

After the installation perform the update of the operating system:

```
$ su -
$ yum update
```

Update /etc/hosts so that the database host knows the other machines in the network:

```
$ su -
$ gedit /etc/hosts
```

Contents of /etc/hosts:

192.168.1.100	hrd-mt-dw.v.nexus	hrd-mt-dw
192.168.1.101	hrd-mt-h01.v.nexus	hrd-mt-h01
192.168.1.102	hrd-mt-h02.v.nexus	hrd-mt-h02
192.168.1.103	hrd-mt-h03.v.nexus	hrd-mt-h03
192.168.1.104	hrd-mt-h04.v.nexus	hrd-mt-h04

For this exercise, we will disable the firewall to facilitate the communication between the different machines in the network:

```
$ su -
$ systemctl stop firewalld
$ systemctl disable firewalld
```

In order to deploy the cluster via Ambari, the Hadoop management tool used by Hortonworks Data Platform, we need to install the Network Time Protocol Daemon service (NTPD).

Install NTP service:

```
$ su -
$ yum install ntp
```


Start the NTPD service and set it to launch automatically upon boot:

```
$ service ntpd start
$ chkconfig ntpd on
```

Ambari, the tool responsible for the cluster installation and deployment, recommends the following maximum open file descriptors configuration:

```
$ ulimit -Sn
$ ulimit -Hn

# if the output is not greater than 10000, run the following command:
$ ulimit -n 10000
```

The definition of these limits should also be present in the shell environment of the user that will execute Ambari. So, the following command is to be added to the '.bashrc' file:

```
ulimit -n 10000
```

Set Up password-less SSH

In hrd-mt-h01:

Generate the key:

```
$ ssh-keygen
```

Create .ssh dir in each machine:

```
$ mkdir /root/.ssh
```

Copy the key from hrd-mt-h01 to hrd-mt-h02:

```
$ scp /root/.ssh/id_rsa.pub root@hrd-mt-h02:/root/.ssh
```

In hrd-mt-h01 and hrd-mt-h02:

Add the SSH public key to the 'authorized_keys' file on each of the target hosts:

```
$ cat /root/.ssh/id_rsa.pub >> /root/.ssh/authorized_keys
```

In hrd-mt-h01:

Validate if each host is accessible from hrd-mt-h01:

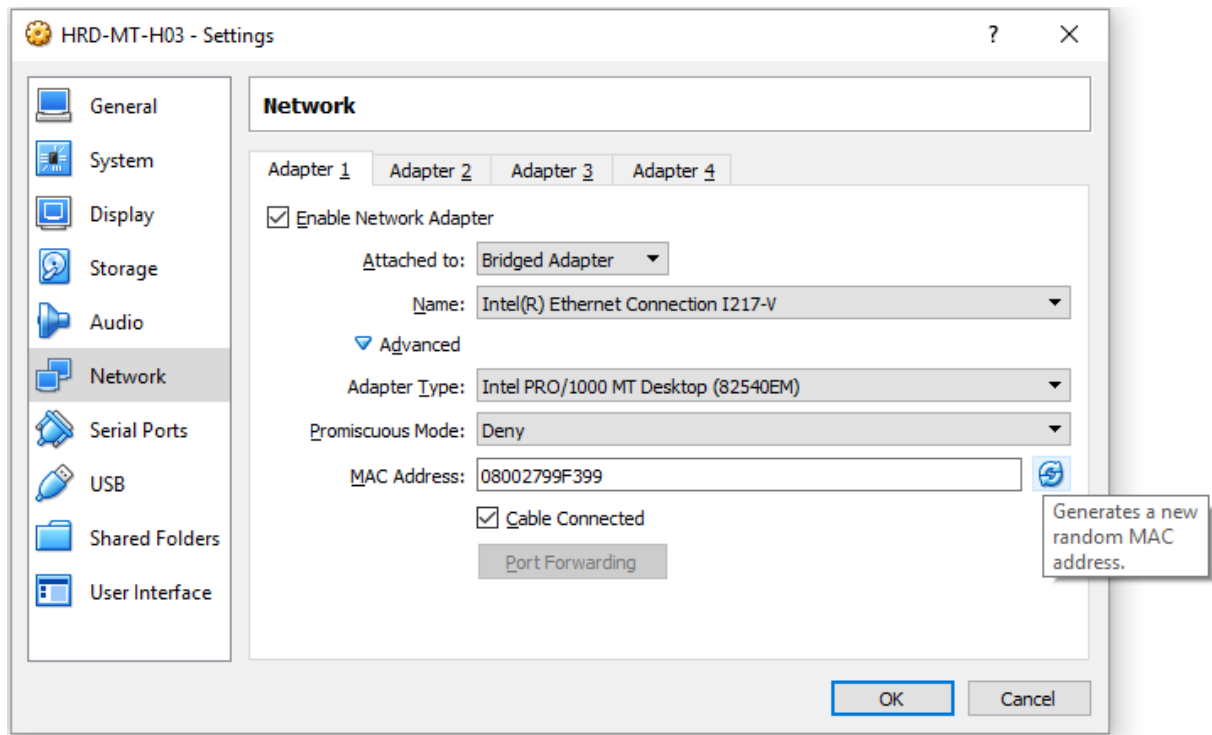
```
$ ssh root@hrd-mt-h02.v.nexus
$ ssh root@hrd-mt-h02.v.nexus
```

The configuration is finished for hrd-mt-h01 and hrd-mt-h02. Shutdown hrd-mt-h02 so that it can be cloned and originate hrd-mt-h03 and hrd-mt-h04.

Annex B.4. NODE CLONING

With the node hrd-mt-h02 powered down, copy its virtual disk to be assigned to hrd-mt-h03 and hrd-mt-h04 virtual machines or simply clone the entire virtual machine twice to create the remaining nodes. Note that this operation can be performed as many times as needed if we just want to add more nodes beyond the four that are the scope of this guide.

After the clone creation, reinitialize the MAC address of the network cards in each of the new nodes that were just created.



Start each of the new nodes in order to adjust some configurations. The following actions are to be performed in each of the new machines.

Change the hostname to the corresponding new name (from hrd-mt-h02 to hrd-mt-hnn):

```
$ vi /etc/hostname
```

Generate new UUIDs to the cloned machines (one to each):

```
$ uuidgen enp0s3
```

With the new UUIDs update the network configuration by replacing the UUID and the IP address:

```
$ vi /etc/sysconfig/network-scripts/ifcfg-enp0s3
```

At this stage, all the nodes that will be a part of the cluster are fully configured and we can begin with the cluster installation, deployment and configuration.

Annex B.5. CLUSTER PRE-INSTALLATION

The first step in installing Hortonworks HDP 2.5 is to install its provisioning and management tool, the Apache Ambari, and in this specific case the version 2.4.

Ambari will be installed on hrd-mt-h01 since this is the node with more resources available and also from this node we will deploy the software to the other nodes.

Starting Ambari installation

1. Login into hrd-mt-h01 as root.

2. Download the Ambari repository file to a directory in the installation host:

```
$ wget -nv http://public-repo-1.hortonworks.com/ambari/centos7/2.x/updates/2.4.1.0/ambari.repo -O /etc/yum.repos.d/ambari.repo
```

3. Download the Ambari repository file to a directory in the installation host:

```
$ yum repolist
```

4. Install the Ambari server. This also installs the default PostgreSQL Ambari database:

```
$ yum install ambari-server
```

Setup the Ambari server

Execute the Ambari setup:

```
$ ambari-server setup
```

Answer to all the prompts with the default option.

Start the Ambari server

Execute the Ambari server:

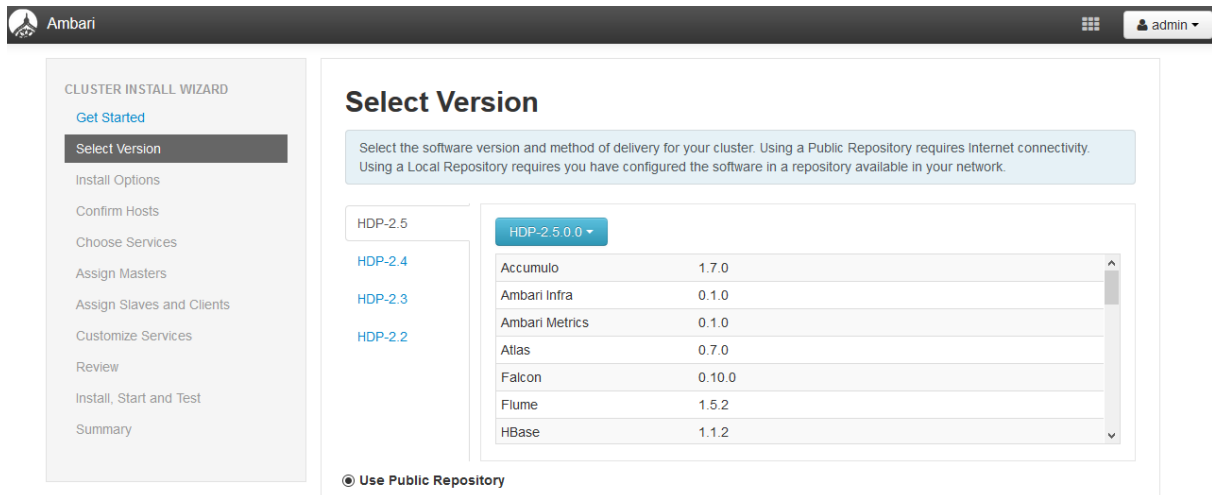
```
$ ambari-server start
```

At this point we are ready to begin the installation of Hortonworks Data Platform and the creation of the Hadoop cluster.

Annex B.6. CLUSTER INSTALLATION

From anywhere in the network we can access the Ambari site and begin the cluster installation.

1. Log in into the Ambari server:
URL: <http://hrd-mt-h01:8080>
Credentials: admin/admin
2. Launch Install Wizard
3. Name cluster: HRD
4. Select version: HDP 2.5.

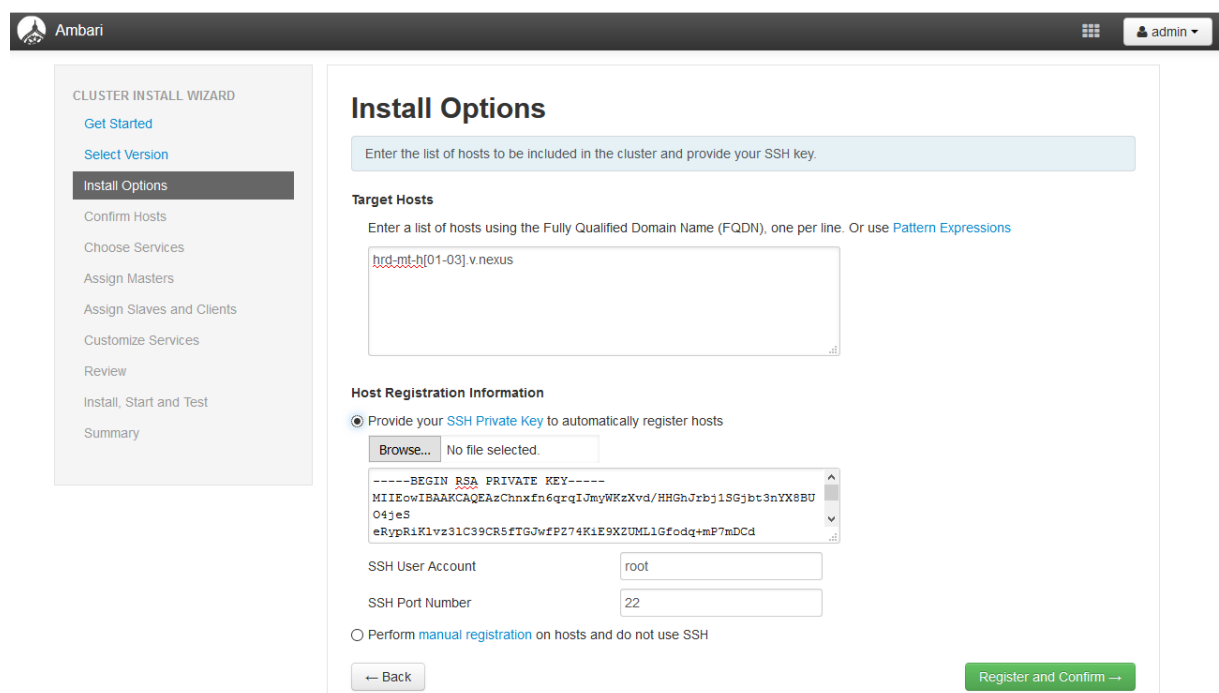


The screenshot shows the Ambari web interface at the 'Select Version' step of the Cluster Install Wizard. The left sidebar contains a navigation menu with options: Get Started, Select Version (highlighted), Install Options, Confirm Hosts, Choose Services, Assign Masters, Assign Slaves and Clients, Customize Services, Review, Install, Start and Test, and Summary. The main content area is titled 'Select Version' and includes a note about public vs. local repositories. A dropdown menu shows 'HDP-2.5' selected. Below it, a table lists available services and their versions:

Service	Version
Accumulo	1.7.0
Ambari Infra	0.1.0
Ambari Metrics	0.1.0
Atlas	0.7.0
Falcon	0.10.0
Flume	1.5.2
HBase	1.1.2

At the bottom, the 'Use Public Repository' radio button is selected.

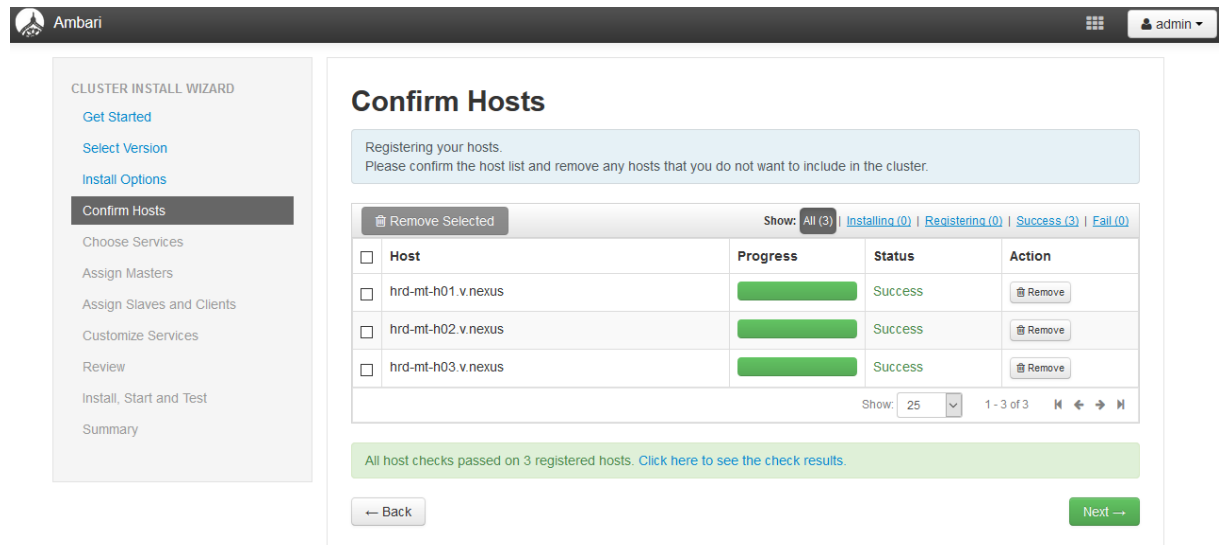
5. Install options:
Add the hosts to where you want to deploy the cluster and to do that use the SSH Private Key previously created and installed (/root/.ssh/id_rsa).



The screenshot shows the Ambari web interface at the 'Install Options' step. The left sidebar is the same as the previous screen, with 'Install Options' highlighted. The main content area is titled 'Install Options' and includes a text box for 'Enter the list of hosts to be included in the cluster and provide your SSH key.' Below this, the 'Target Hosts' section has a text box containing 'hrd-mt-h[01-03].v.nexus'. The 'Host Registration Information' section has the 'Provide your SSH Private Key' radio button selected. A 'Browse...' button is shown with 'No file selected.' Below it, a text box displays the contents of the SSH private key file, starting with '-----BEGIN RSA PRIVATE KEY-----'. The 'SSH User Account' field is set to 'root' and the 'SSH Port Number' field is set to '22'. At the bottom, the 'Perform manual registration' radio button is unselected. There are 'Back' and 'Register and Confirm' buttons.

6. Confirm the hosts that will be part of the cluster:

Note that even though we have prepared four nodes to be part of the cluster, we will only install the first three at this stage. The fourth node can be added with the cluster already up and running.

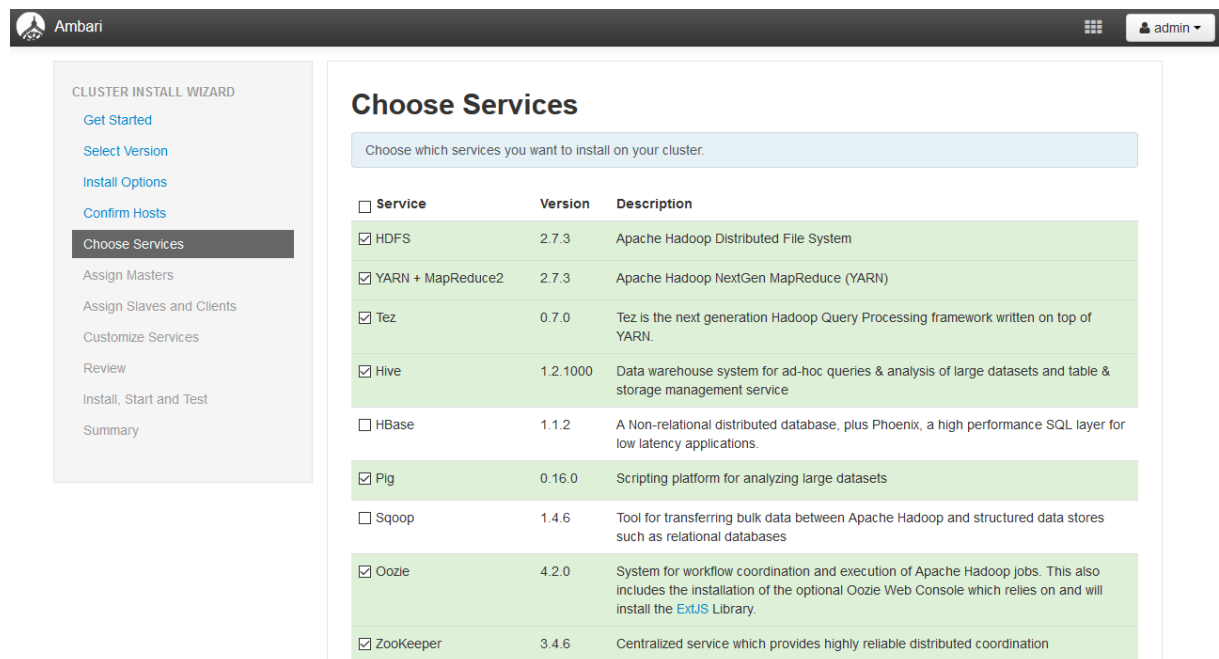


The screenshot shows the Ambari web interface during the 'Confirm Hosts' step of the cluster installation wizard. The left sidebar contains the 'CLUSTER INSTALL WIZARD' menu with options: Get Started, Select Version, Install Options, Confirm Hosts (selected), Choose Services, Assign Masters, Assign Slaves and Clients, Customize Services, Review, Install, Start and Test, and Summary. The main content area is titled 'Confirm Hosts' and includes a message: 'Registering your hosts. Please confirm the host list and remove any hosts that you do not want to include in the cluster.' Below this is a table with columns: Host, Progress, Status, and Action. The table lists three hosts: hrd-mt-h01.v.nexus, hrd-mt-h02.v.nexus, and hrd-mt-h03.v.nexus, all with a progress bar and a 'Success' status. Each host has a 'Remove' button in the Action column. At the bottom of the table, there is a 'Show' dropdown set to '25' and a pagination indicator '1 - 3 of 3'. Below the table, a green message bar states: 'All host checks passed on 3 registered hosts. Click here to see the check results.' At the bottom of the main area are 'Back' and 'Next' buttons.

Host	Progress	Status	Action
<input type="checkbox"/> hrd-mt-h01.v.nexus	<div></div>	Success	Remove
<input type="checkbox"/> hrd-mt-h02.v.nexus	<div></div>	Success	Remove
<input type="checkbox"/> hrd-mt-h03.v.nexus	<div></div>	Success	Remove

7. Choose the services that will be installed in the cluster:

On this installation, besides the basic services necessary for the cluster like HDFS and Yarn, we are going to install Hive and Tez. Service selection at this point is not important since services can be added or removed at any given time.

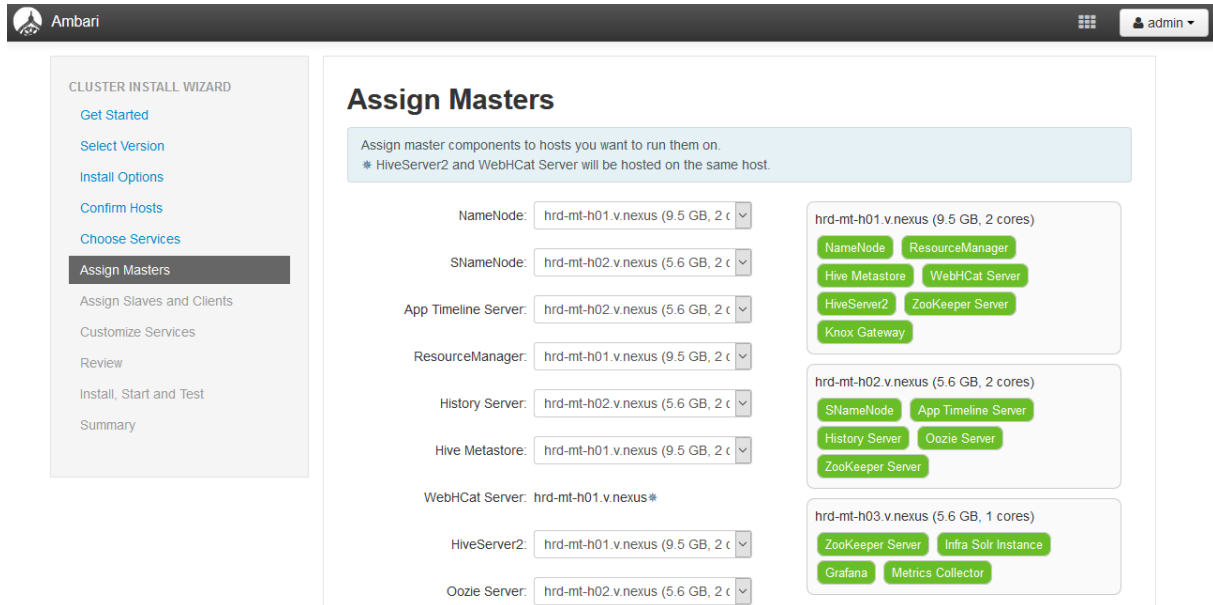


The screenshot shows the Ambari web interface during the 'Choose Services' step of the cluster installation wizard. The left sidebar contains the 'CLUSTER INSTALL WIZARD' menu with options: Get Started, Select Version, Install Options, Confirm Hosts, Choose Services (selected), Assign Masters, Assign Slaves and Clients, Customize Services, Review, Install, Start and Test, and Summary. The main content area is titled 'Choose Services' and includes a message: 'Choose which services you want to install on your cluster.' Below this is a table with columns: Service, Version, and Description. The table lists several services: HDFS (checked), YARN + MapReduce2 (checked), Tez (checked), Hive (checked), HBase (unchecked), Pig (checked), Sqoop (unchecked), Oozie (checked), and ZooKeeper (checked). Each service has a checkbox in the Service column. The table is styled with alternating light green and white rows.

Service	Version	Description
<input checked="" type="checkbox"/> HDFS	2.7.3	Apache Hadoop Distributed File System
<input checked="" type="checkbox"/> YARN + MapReduce2	2.7.3	Apache Hadoop NextGen MapReduce (YARN)
<input checked="" type="checkbox"/> Tez	0.7.0	Tez is the next generation Hadoop Query Processing framework written on top of YARN.
<input checked="" type="checkbox"/> Hive	1.2.1000	Data warehouse system for ad-hoc queries & analysis of large datasets and table & storage management service
<input type="checkbox"/> HBase	1.1.2	A Non-relational distributed database, plus Phoenix, a high performance SQL layer for low latency applications.
<input checked="" type="checkbox"/> Pig	0.16.0	Scripting platform for analyzing large datasets
<input type="checkbox"/> Sqoop	1.4.6	Tool for transferring bulk data between Apache Hadoop and structured data stores such as relational databases
<input checked="" type="checkbox"/> Oozie	4.2.0	System for workflow coordination and execution of Apache Hadoop jobs. This also includes the installation of the optional Oozie Web Console which relies on and will install the ExtJS Library.
<input checked="" type="checkbox"/> ZooKeeper	3.4.6	Centralized service which provides highly reliable distributed coordination

8. Assign the masters:

The distribution of services, by the nodes, should be performed according to the resources of each node and bearing in mind the interdependences among the services. Balancing the services by the nodes can and should be done at a later time according to the workload in each node when we have the cluster running the intended processes.



Assign Masters

Assign master components to hosts you want to run them on.
* HiveServer2 and WebHCat Server will be hosted on the same host.

NameNode: hrd-mt-h01.v.nexus (9.5 GB, 2 c)
 SNameNode: hrd-mt-h02.v.nexus (5.6 GB, 2 c)
 App Timeline Server: hrd-mt-h02.v.nexus (5.6 GB, 2 c)
 ResourceManager: hrd-mt-h01.v.nexus (9.5 GB, 2 c)
 History Server: hrd-mt-h02.v.nexus (5.6 GB, 2 c)
 Hive Metastore: hrd-mt-h01.v.nexus (9.5 GB, 2 c)
 WebHCat Server: hrd-mt-h01.v.nexus *
 HiveServer2: hrd-mt-h01.v.nexus (9.5 GB, 2 c)
 Oozie Server: hrd-mt-h02.v.nexus (5.6 GB, 2 c)

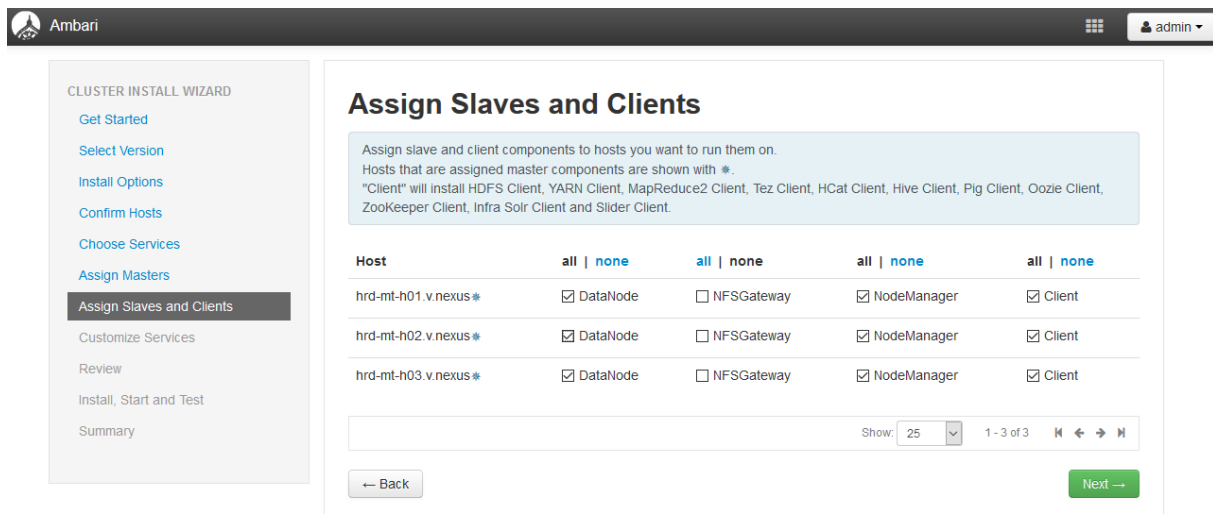
hrd-mt-h01.v.nexus (9.5 GB, 2 cores)
 NameNode, ResourceManager, Hive Metastore, WebHCat Server, HiveServer2, ZooKeeper Server, Knox Gateway

hrd-mt-h02.v.nexus (5.6 GB, 2 cores)
 SNameNode, App Timeline Server, History Server, Oozie Server, ZooKeeper Server

hrd-mt-h03.v.nexus (5.6 GB, 1 cores)
 ZooKeeper Server, Infra Solr Instance, Grafana, Metrics Collector

9. Assigning Clients and Slaves:

Since every node will be a DataNode they will all be slaves. The selection of a node to install the client software depends if we intend to execute jobs of a given service from a specific node. This configuration can also be changed after the cluster is installed and running.



Assign Slaves and Clients

Assign slave and client components to hosts you want to run them on.
Hosts that are assigned master components are shown with *.
"Client" will install HDFS Client, YARN Client, MapReduce2 Client, Tez Client, HCat Client, Hive Client, Pig Client, Oozie Client, ZooKeeper Client, Infra Solr Client and Slider Client.

Host	all none	all none	all none	all none
hrd-mt-h01.v.nexus *	<input checked="" type="checkbox"/> DataNode	<input type="checkbox"/> NFSGateway	<input checked="" type="checkbox"/> NodeManager	<input checked="" type="checkbox"/> Client
hrd-mt-h02.v.nexus *	<input checked="" type="checkbox"/> DataNode	<input type="checkbox"/> NFSGateway	<input checked="" type="checkbox"/> NodeManager	<input checked="" type="checkbox"/> Client
hrd-mt-h03.v.nexus *	<input checked="" type="checkbox"/> DataNode	<input type="checkbox"/> NFSGateway	<input checked="" type="checkbox"/> NodeManager	<input checked="" type="checkbox"/> Client

Show: 25 1 - 3 of 3

← Back Next →

10. Customize Services:

Each service has its own configuration and they can be set at this stage. In this installation, we are using all the default configurations at this stage. Specific configurations to certain services will be applied after we collect some information about the cluster performance while running jobs.

11. Install, Start and Test:

In this step Ambari deploys the selected services to the corresponding nodes according to our previous configurations.

Ambari

admin

CLUSTER INSTALL WIZARD

- Get Started
- Select Version
- Install Options
- Confirm Hosts
- Choose Services
- Assign Masters
- Assign Slaves and Clients
- Customize Services
- Review
- Install, Start and Test**
- Summary

Install, Start and Test

Please wait while the selected services are installed and started.

3 % overall

Show: All (3) | In Progress (3) | Warning (0) | Success (0) | Fail (0)

Host	Status	Message
hrd-mt-h01.v.nexus	3%	Waiting to install DataNode
hrd-mt-h02.v.nexus	3%	Waiting to install App Timeline Server
hrd-mt-h03.v.nexus	3%	Waiting to install DataNode

3 of 3 hosts showing - [Show All](#)

Show: 25 1 - 3 of 3

Next →

When this step is complete, our cluster is finally installed and running.

Annex B.7. ISSUES FOUND

Ambari server does not start at system startup

After the installation, the Ambari server is not being executed automatically at system startup. This known issue (AMBARI-14526²³) can be solved by adding the following code marked in bold to the file `'/sbin/ambari-server'`:

```
export ROOT=`dirname $(dirname $SCRIPT_DIR)`
ROOT=`echo $ROOT | sed 's/\$//'`

# RHEL 7.2 places script under /etc/rc.d/init.d, which makes ROOT=/etc
if [ ! -d "$ROOT/usr/lib/ambari-server" ]; then
    export ROOT=`dirname $ROOT`
    ROOT=`echo $ROOT | sed 's/\$//'`
fi

# If for any reason the ROOT is still not correct fail here
if [ ! -d "$ROOT/usr/lib/ambari-server" ]; then
    echo "Can't locate Ambari lib folder under: $ROOT/usr/lib/ambari-server"
    exit 1
fi

export PATH=$ROOT/usr/lib/ambari-server/*:$PATH:/sbin:/usr/sbin
export AMBARI_CONF_DIR=$ROOT/etc/ambari-server/conf
```

DataNodes fail to install

This is a HDP known issue (BUG-41308²⁴) and can be easily solved by rollbacking the snappy version in the machine hosting the DataNode with the following instructions:

```
$ yum remove snappy
$ yum install snappy-devel
```

Zeppelin Notebook does not start

After the cluster installation, the Zeppelin Notebook service failed to start due to permission issues. To fix this perform the following instructions in the node hosting the service:

```
$ mkdir -p /var/run/zeppelin-notebook
$ chown -R zeppelin:zeppelin /var/run/zeppelin-notebook
$ mkdir -p /var/run/zeppelin
$ chown -R zeppelin:zeppelin /var/run/zeppelin
```

²³ <https://issues.apache.org/jira/browse/AMBARI-14526> (Accessed on 2016-10-24)

²⁴ https://docs.hortonworks.com/HDPDocuments/Ambari-2.1.0.0/bk_releasenotes_ambari_2.1.0.0/content/ambari_relnotes-2.1.0.0-known-issues.html (Accessed on 2016-10-27)